Computer Science Department

TECHNICAL REPORT

A Linear Time Algorithm for Finding an Ambitus

Bhubaneswar Mishra
Robert E. Tarjan

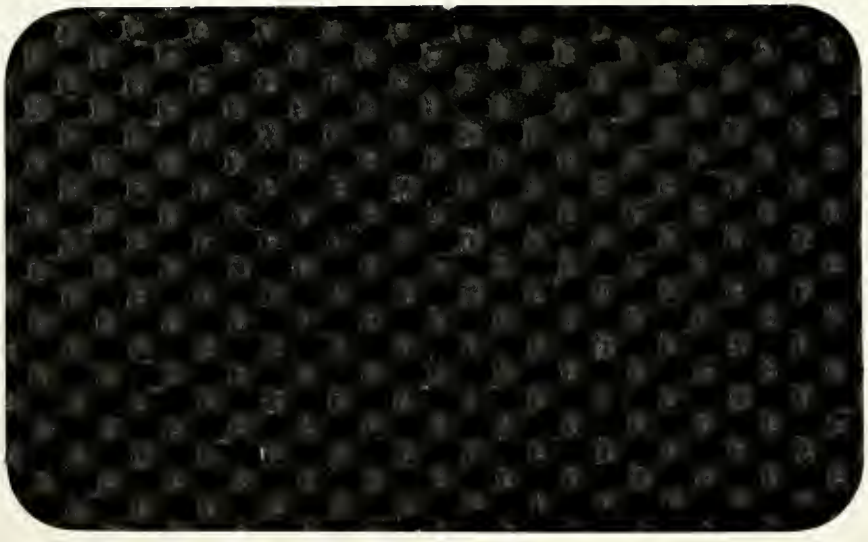
Technical Report 464

August 1989

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012

NYU COMPSCI TR-464
Mishra, Bhubaneswar
A linear time algorithm for finding an ambitus.
c.1



A Linear Time Algorithm for Finding an Ambitus

Bhubaneswar Mishra
Robert E. Tarjan

Technical Report 464

August 1989

A Linear Time Algorithm for Finding an Ambitus

B. Mishra

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
719 Broadway
New York, NY 10003

R.E. Tarjan

Department of Computer Science
Princeton University
Princeton, NJ 08544

August 26, 1989

ABSTRACT

We devise a linear time algorithm for finding an ambitus in an undirected graph. An ambitus is a cycle in a graph containing two distinguished vertices such that certain different groups of bridges (called B^P-, B^Q- and B^{PQ}-bridges) satisfy the property that a bridge in one group does not interlace with any bridge in the other groups. Thus, an ambitus allows the graph to be cut into pieces, where, in each piece, certain graph properties may be investigated independently and recursively, and then the pieces can be pasted together to yield information about these graph properties in the original graph. In order to achieve a good time-complexity for such an algorithm employing the divide-and-conquer paradigm, it is necessary to find an ambitus quickly. We also show that, using ambitus, linear-time algorithms can be devised for abiding-path-finding and nonseparable-induced-cycle-finding problems.

Contents

1 Introduction 1

2 Preliminaries 2
- 2.1 Graph Theoretic Terminology . . . 2
- 2.2 Bridge and Ambitus . . . 3
- 2.3 Properties of Bridges and Nonseparable Components . . . 7

3 Existence of an Ambitus 8

4 Sketch of the Algorithm 11
- 4.1 The Algorithm . . . 11
- 4.2 The Correctness of the Algorithm Find-Ambitus . . . 13

5 Implementation: Path-Finding 18
- 5.1 Building the Palm-Tree . . . 18
- 5.2 Θ-Graph and its Bridges. . . . 24
- 5.3 Using the Palm-Tree . . . 24

6 Implementation: The Auxiliary Data Structure 30
- 6.1 The Data Structure . . . 30
- 6.2 Building the Forest structure . . . 33
- 6.3 Using the Data Structure . . . 37

7 The Timing-Analysis of the Algorithm Find-Ambitus 41

8 Abiding Path and Nonseparating Induced Cycle 43

9 Acknowledgement 44

1 Introduction

The concept of an ambitus was first introduced in [6] and [7], in the process of devising an efficient divide-and-conquer algorithm for the All-Bidirectional-Edge problem. An ambitus is a cycle in a graph containing two distinguished vertices such that certain different groups of bridges (called B^P-, B^Q- and B^{PQ}-bridges) satisfy the property that a bridge in one group avoids (*i.e.*, does not interlace with) every bridge in the other groups. (See section 2, for a more formal definition of ambitus.) Thus, an ambitus allows the graph to be cut into pieces, where, in each piece, certain graph properties may be investigated independently and recursively, and then the pieces can be pasted together to yield information about these graph properties in the original graph. In order to achieve a good time-complexity for such an algorithm employing the divide-and-conquer paradigm, it is necessary to find an ambitus *quickly*, *i.e.*, in $O(|E| + |V|)$ time. Such an algorithm first appears in [7] and yields a time complexity of $O(|E| \cdot |V|)$ for the algorithm to find all bidirectional edges of an undirected graph.

In many respects, this algorithm is a generalization of the planarity testing algorithm, due to Hopcroft and Tarjan (see [4]). Like their planarity testing algorithm, the ambitus-finding algorithm needs to decompose the graph in to a set of internally-vertex disjoint subpaths, although the decomposition needed for our purpose has to be somewhat different. Unlike their planarity testing algorithm, the ambitus-finding algorithm cannot traverse the subpaths in an order, known *a priori*; rather the order, in which the subpaths are traversed, has to be determined dynamically. For this purpose, we have developed a novel data structure that maintains a set of (integral) intervals and supports a fast FIND-AND-UPDATE operation that detects the interval corresponding to a subpath to be visited next and updates the set of intervals, appropriately. This data structure has a good *amortized* time-complexity, and may be of independent of interest.

Subsequently, many other researchers have developed new algorithms either for ambitus-finding in special classes of graphs or for certain other closely related concepts in general graphs. Two such related concepts are *abiding paths* and *nonseparating induced cycles*. In section 8, we present linear time algorithms for abiding paths in nonseparable graphs and nonseparating induced cycles in triconnected graphs. The abiding paths have to be found in a 'divide' step for an algorithm, due to Ohtsuki ([8]), for the two-vertex-disjoint-paths problem. Nonseparating induced cycles need to be found repeatedly (at most $|V|$ times) in an algorithm, due to Cheriyan and Maheswari ([1]), that finds three independent spanning trees rooted at a distinguished vertex, in a triconnected graph.

In [5], Krishnan, Pandu Rangan and Sheshadri have developed a simple linear time algorithm to find an ambitus in a planar graph. Although their algorithm is simple and elegant, the techniques employed do not generalize to other classes of graphs. In [11], R. Sundar has developed an $O(|E| + |V| \log |V|)$ time algorithm to find an abiding path in a nonseparable graph. This algorithm can be used for the ambitus-finding problem, if the graph is suitably modified. However, such an algorithm has a linear-time behavior, only if the graph is dense. In [1], Cheriyan and Maheswari have developed a linear-time algorithm to find a nonseparating induced cycle in a triconnected graph. With suitable modifications to certain basic steps in the algorithm, a different linear-time algorithm for the ambitus-finding problem can be devised.

The paper is organized as follows: In section 2, we define some graph theoretic terminology

and introduce other key concepts required in the paper. In section 3, we demonstrate that an ambitus always exists in a nonseparable graph, and an ambitus can be found by a naïve algorithm of complexity. $O(|E| \cdot |V|)$. In section 4, we provide a sketch of the main algorithm, and in the subsequent two sections, we provide the implementation details, in order to guarantee a linear-time behavior. In the last section, we give two simple applications of the ambitus-finding algorithm.

2 Preliminaries

In this section, we define some graph theoretic terminology and introduce other key concepts required in the paper. Most important among these are the terms: *bridge*, *residual path*, *cross-cut* and *ambitus*. The definitions are similar to those used in the context of Tutte's Theorem on Hamiltonian circuits in 4-connected planar graphs given in Tutte(1977)[13], those in the planarity-testing algorithm of Hopcroft and Tarjan(1978)[4] or those in connection with the four-color problem as presented in Ore[9]. The reader with some familiarity with graph theory and related algorithms is advised to skip this section and refer it only when (s)he comes across an unfamiliar term.

2.1 Graph Theoretic Terminology

A *graph* $G = (V, E)$ consists of set V of *vertices*, a set E of *edges*, and a relation of *incidence*, which associates with each edge either one or two vertices called its *ends*. We write $V(G)$ and $E(G)$ to denote the vertices and the edges of the graph G, respectively. An edge is called a *link* or a *loop* according as the number of its ends is two or one. Two or more links with the same pair of ends are said to constitute a *multi-link*.

All graphs considered here are assumed to be *finite* (the sets $V(G)$ and $E(G)$ are both finite) and *strict* (without a loop or multi-link).

Either the edges are ordered pairs $\langle u, v\rangle$ of distinct vertices (the graph is *directed*) or the edges are unordered pairs $[u, v]$ of distinct vertices (the graph is *undirected*). If $[u, v]$ is an undirected edge, u and v are *adjacent*. If $\langle u, v\rangle$ is a directed edge, u is a *predecessor* of v (respectively, v is a *successor* of u), sometimes denoted by, $u \rightarrow v$ (respectively, $v \leftarrow u$).

A graph $G_1 = (V_1, E_1)$ is a *subgraph* of G, if $V_1 \subseteq V(G)$, $E_1 \subseteq E(G)$, and each edge of G_1 has the same ends in G_1 as in G. If G_1 is a subgraph of G, other than G itself, then G_1 is a *proper* subgraph of G. If $V_1 = V(G)$ then G_1 is said to be a *spanning subgraph* of G. A *vertex of attachment* of G_1 in G is a vertex of G_1 that is incident in G with some edge not belonging to G_1.

If $E_1 \subseteq E(G)$, let $V(E_1)$ be the set of all vertices v of G such that v is incident with a member of E_1, *i.e.*,

$$V(E_1) = \{v \in V(G) \mid \exists_{u \in V(G)} [u, v] \in E_1\}.$$

Then the subgraph $\langle E_1\rangle = (V(E_1), E_1)$ is the *reduction* of G to E_1. Similarly, if $V_1 \subseteq V(G)$, let $E(V_1)$ be the set of all edges of G having both ends in V_1,*i.e.*,

$$E(V_1) = \{[u, v] \in E(G) \mid u, v \in V_1\}.$$

Then the subgraph $\langle V_1 \rangle = (V_1, E(V_1))$ is the subgraph of G *induced* by V_1.

An undirected graph is *connected* if there is a path connecting every pair of vertices and *disconnected* otherwise. The maximal connected subgraphs of G are its *connected components* or simply, *components*.

A *path* of *length* k from u to v in G is a sequence of vertices $(u =)\ u_0, u_1, \ldots, u_k\ (= v)$ such that $\langle u_i, u_{i+1} \rangle \in E$ for $0 \leq i < k$. (Sometimes denoted by $u \xrightarrow{*} v$.) The path *contains* the edges $\langle u_i, u_{i+1} \rangle$ for $0 \leq i < k$ as well as vertices u_i for $0 \leq i \leq k$. The vertices u and v are called the *ends* of the path P. All other vertices of the path (*i.e.*, u_i's for $0 < i < k$) are the *internal vertices* of the path.

If $0 \leq i \leq j \leq k$, then the sequence of vertices, $u_i, u_{i+1}, \ldots, u_j$ is a *subpath* of the path from u to v. If P is a path from u to v, $u = u_0, u_1, \ldots, u_k = v$, and $0 \leq i \leq j \leq k$ then the subpath from u_i to u_j, including both u_i and u_j is represented by $P[u_i; u_j]$; the subpath excluding u_i but including u_j, by $P]u_i; u_j]$; the subpath including u_i but excluding u_j, by $P[u_i; u_j[$ and the subpath excluding both u_i and u_j, by $P]u_i; u_j[$. If $P = u_0, u_1, \ldots, u_{k-1}, u_k$ is a path from u_0 to u_k, then the *reversal* of the path P is $P^{\mathrm{R}} = u_k, u_{k-1}, \ldots, u_1, u_0$. If $P_1 = u_0, u_1, \ldots, u_i$ and $P_2 = u_i, u_{i+1}, \ldots, u_k$ are two paths then the *concatenation* of P_1 and P_2 is $P_1 * P_2 = u_0, u_1, \ldots, u_i, u_{i+1}, \ldots, u_k$.

The path is *simple* if $u_0, \ldots, u_k$ are distinct (except possibly $u_0 = u_k$) and the path is a *cycle* if $u_0 = u_k$. By convention there is a path of no edges from every vertex to itself (*null path*), but a cycle must contain at least two edges. Two simple paths P_1 and P_2 are said to be *vertex-disjoint*, if the vertices of P_1 and P_2 are mutually distinct; *internally vertex-disjoint*, if the internal vertices of P_1 and P_2 are mutually distinct; and *edge-disjoint*, if the edges of P_1 and P_2 are mutually distinct.

A connected graph is said to have a *separation vertex* v (also called an articulation point) if there exist vertices a and b, $a \neq v$ and $b \neq v$, such that all the paths connecting a and b pass through v. A graph which has a separation vertex is called *separable*, and one which has none is called *nonseparable* (also called biconnected). Let $V' \subseteq V$. The induced graph $G' = \langle V' \rangle$ is called a *nonseparable component* (also called a biconnected component) if G' is nonseparable and if for every larger $V'', V' \subset V'' \subseteq V$, the induced graph $G'' = \langle V'' \rangle$ is separable.

2.2 Bridge and Ambitus

In this section we introduces the notion of bridges for cycles in general graphs. The term 'bridge' is taken from Tutte(1977)[13], which also contains a rather complete survey of bridge theory in both general and planar graphs. The equivalent terms for bridge, in some older literature, are 'component *mod* J', 'J-component'[12] and 'Gespinst'[1][10].

Definition 2.1 BRIDGES.[Tutte]

Let J be a fixed subgraph of G. A subgraph G_1 of G is said to be *J-detached* in G, if all its vertices of attachment are in J. We define a *bridge* of J in G as any subgraph B that satisfies the three following conditions:

1. *B is not a subgraph of J.*

[1] Gespinst is the German word for 'cobweb'.

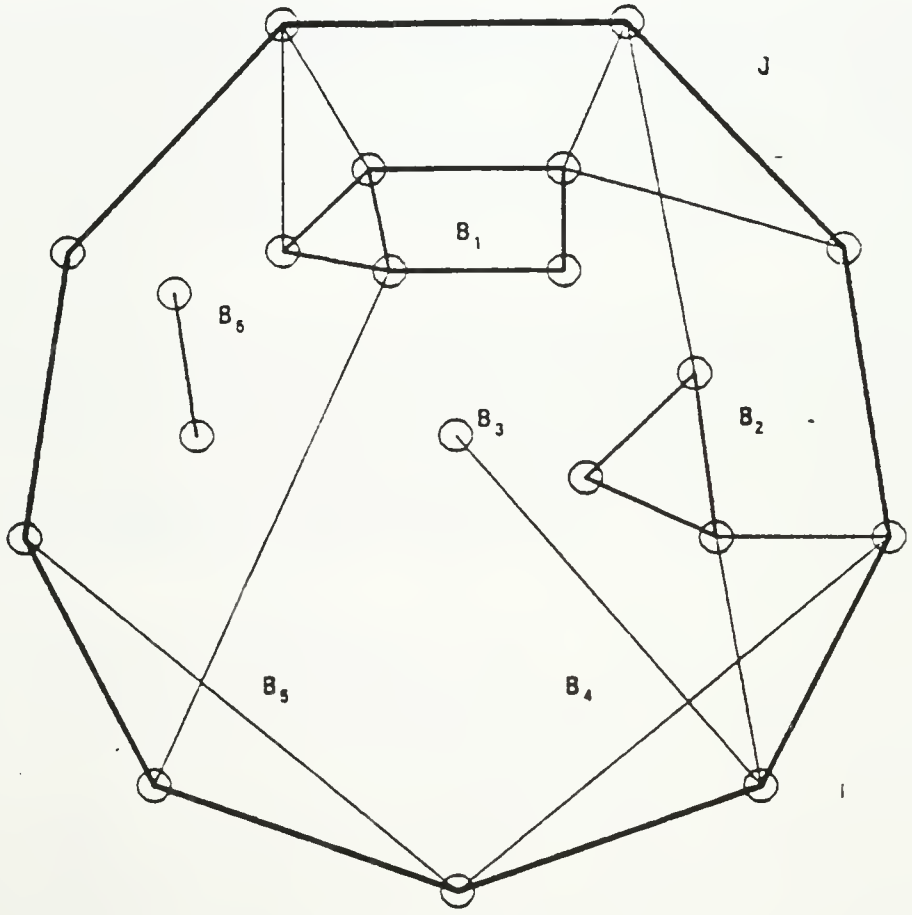


Figure 1: Bridges of J.

2. *B is J-detached in G.*

3. *No proper subgraph of B satisfies both (1) and (2).*

The set of vertices of attachment of a bridge B of a subgraph J in G is denoted by $W(G,B) = \{v_0, v_1, \ldots, v_{k-1}\}$. $w(G,B)$ will stand for $|W(G,B)|$. If $w(G,B) = n$ then B is said to be an *n-bridge*. □

Definition 2.2 DEGENERATE AND PROPER BRIDGES. NUCLEUS OF A BRIDGE.

An edge $e = [u,v]$ of G not belonging to J but having both ends in J is referred to as a *degenerate* bridge.

Let G^- be the graph derived from G by deleting the vertices of J and all their incident edges. Let C be any component of G^-. Let B be the subgraph of G obtained from C by adjoining to it each edge of G having one end in C and one in J, and adjoining also the ends in J of all such edges. The subgraph B satisfies the conditions to be a bridge. Such a bridge is called *proper*. The component C of G^- is the *nucleus* of B. □

Remark 2.3 By a Theorem due to Tutte, if B is any bridge of a subgraph J of G then B is either degenerate or proper. Hence using the above definition of a bridge and the theorem, we can give a linear time algorithm to find all bridges of a subgraph J of G.

In the remaining portion of this paper, we will assume the subgraph J of G is a cycle in G. □

Example 2.4 In the Figure 1, we give an example of bridges of a cycle J. In this example, bridges B_1, B_2, B_3 and B_6 are proper bridges, and B_4 and B_5 are degenerate bridges.

Definition 2.5 RESIDUAL PATHS.

Let the vertices of attachment of a bridge B of a cycle J in G, be $W(G,B) = \{v_0, v_1, \ldots, v_{k-1}\}$ and let $v_0, v_1, \ldots, v_{k-1}$ be their enumeration in their cyclic order on J. The vertices of attachment dissect J into k subpaths $L_0, L_1, \ldots, L_{k-1}$ such that $L_j = J[v_j; v_{j+1(\mathrm{mod}k)}]$. These subpaths are called the *residual paths* of B in J. □

Definition 2.6 RELATIONS BETWEEN BRIDGES.

Let B_1 and B_2 be two distinct bridges of a cycle J of G.

- We say B_1 *avoids* B_2 if and only if one of the following two conditions is satisfied:

  1. $w(G, B_1) \leq 1$ or $w(G, B_2) \leq 1$.
  2. *All the vertices of attachment of* B_1 *are contained in a single residual path* L *of* B_2.

- If B_1 and B_2 do not avoid one another we say that they *overlap*.
- If there exist two vertices of attachment x_1 and x_2 of B_1 and two vertices of attachment y_1 and y_2 of B_2, all four distinct, such that x_1 and x_2 separate y_1 and y_2 in the cycle J, then we say that they *interlace*.
- If B_1 and B_2 have exactly the same set of vertices of attachment we say that they are *equivalent*.

□

Example 2.7 In the example shown in Figure 1, the bridges B_1 and B_2 interlace, where as, the bridge B_1 avoids B_4. Also notice that the bridge B_3 avoids every other bridge.

Definition 2.8 CROSS-CUTS.

Let J be a cycle of the graph G. A path N in G avoiding J but having its two ends x and y in J is called a *cross-cut* of J between x and y.

The vertices x and y dissect J into internally vertex disjoint paths L_1 and L_2. We also call them the *residual paths* in J of the cross-cut N.

Let N_1 and N_2 be two cross-cuts of J between x_1 and x_2, and y_1 and y_2, respectively in G. We say that N_1 and N_2 *interlace*, if x_1 and x_2 separate y_1 and y_2 in the cycle J. □

Definition 2.9 PATHS P AND Q.

Let G be an undirected graph with two distinguished vertices s and t and let J be a cycle of the graph containing the vertices, s and t. Let the vertices of J be ordered in a clock-wise cyclic order starting with the vertex s. A subpath $J[a;b] = (a =)u_0, u_1, \ldots, u_{k-1}, u_k(= b)$ denotes the *unique* subpath of J in which u_{i-1} precedes u_i in the clock-wise cyclic order (for $1 \leq i \leq k$).

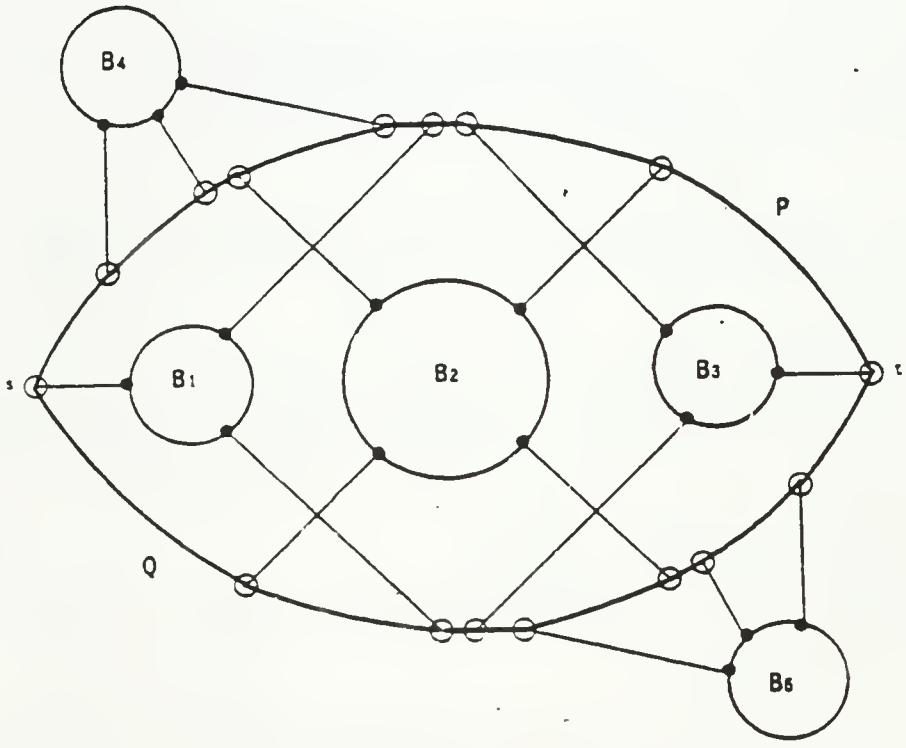


Figure 2: B^{PQ}-, B^{P}- and B^{Q}-bridges of P and Q.

The vertices, s and t dissect the cycle J into two internally vertex-disjoint paths: $P[s;t]$, where $P = J[s;t]$ and its complementary subpath in J, $Q[s;t]$, where $Q^{\mathrm{R}} = J[t;s]$. Clearly P and Q are internally vertex disjoint.

The vertices of P are ordered according to the cyclic order, and vertices of Q, according to the reverse-cyclic order. A vertex u of P is said to be *to the left* of a vertex v of P, if u precedes v in the cyclic order of J; and u is *strictly to the left* of v, if, in addition, u and v are distinct. On the other hand, a vertex u of Q is said to be *to the left* of a vertex v of Q, if v precedes u in the cyclic order of J; and u is *strictly to the left* of v, if, in addition, u and v are distinct. The relation '*to the right of*' is the inverse of the relation 'to the left of;' and the relation '*strictly to the right of*' is the relation 'to the right of' with additional irreflexivity property. □

Definition 2.10 BRIDGES WITH RESPECT TO THE PATHS.

Let G be an undirected graph with two distinguished vertices s and t with two internally vertex-disjoint paths $P[s;t]$ and $Q[s;t]$, which meet each other only in their end vertices, s and t. We consider three different classes of bridges of interest to us:

- B^{PQ}-BRIDGES: *The set of bridges with* at least one *vertex of attachment on* $P]s;t[$ *and* at least one *vertex of attachment on* $Q]s;t[$.
- B^{P}-BRIDGES: *The set of bridges with* at least one *vertex of attachment on* $P]s;t[$ *and* no *vertex of attachment on* $Q]s;t[$.

- B^Q-BRIDGES: *The set of bridges with* no *vertex of attachment on* $P]s;t[$ *and at least one vertex of attachment on* $Q]s;t[$.

If a bridge B of $J = \{P\} \cup \{Q\}$ in G is not a B^{PQ}-, B^P- or B^Q-bridge then it has only s or t as vertices of attachment. □

Example 2.11 In the figure 2, we show B^{PQ}-, B^P- and B^Q- bridges of the paths P and Q. Bridges B_1, B_2 and B_3 are B^{PQ}-bridges; B_4 is a B^P-bridge and B_5, a B^Q-bridge.

Definition 2.12 AMBITUS.

Let J, P and Q be as in the previous definition. Then J is called an *ambitus* if every B^P- or B^Q-bridge avoids every B^{PQ}-bridge. □

Problem 2.1 AMBITUS-FINDING PROBLEM.

Assume that $G = (V, E)$ is a nonseparable graph containing two distinguished vertices s and t.

Find two internally vertex disjoint paths $P[s;t]$ and $Q[s;t]$ in G such that the cycle $J = P*Q^R$ is an ambitus. □

Notation 2.13 Let G, s, t, $P[s;t]$, $Q[s;t]$ and J be as before. If B is a bridge of the cycle J with at least one vertex of attachment on $P]s;t[$, then the left- and the right-most vertices of attachment of B on $P[s;t]$ are referred to by $s_P(B)$ and $t_P(B)$. Similarly, if B is a bridge of the cycle J with at least one vertex of attachment on $Q]s;t[$, then the left- and the right-most vertices of attachment of B on $Q[s;t]$ are referred to by $s_Q(B)$ and $t_Q(B)$. □

2.3 Properties of Bridges and Nonseparable Components

In this section we present some technical propositions that will be used quite often later on.

Proposition 2.1 (Tutte[13]) *Let x be any edge or vertex of G not belonging to the cycle J. Then x belongs to exactly one bridge of J of G that is degenerate or proper.* □

Proposition 2.2 (Tutte[13]) *Let B be any bridge of a cycle J of G. Then B is either degenerate or proper.* □

Proposition 2.3 (Tutte[13]) *Let x and y be distinct vertices of a bridge B of J in G. Then there is a path N in B whose ends are x and y and which avoids J. N is a cross-cut of J between x and y.* □

Proposition 2.4 (Mishra[7]) *Let y be a vertex of G belonging to some bridge B of J and x be any vertex $x \in G \setminus B$ such that there is a path from x to y in G. Then there is a vertex z on this path that is also a vertex of attachment of B.* □

Proposition 2.5 (Even[2]) *Let x, y and z be three distinct vertices of attachment of a bridge B of J in G. Then there is a vertex v belonging to the nucleus of B for which there are three internally vertex disjoint paths in B: $Y_1[x;v]$, $Y_2[y:v]$ and $Y_3[z;v]$.* □

Remark 2.14 Following Tutte[14], we define a *Y-graph* as the union Y of three paths Y_1, Y_2 and Y_3 which have one end v in common but are otherwise mutually disjoint. We call v the *center* and the paths Y_i's, the *arms* of Y.

Proposition 2.6 (Mishra[7]) *Let G be a nonseparable graph with a cycle J and let $B_1, \ldots, B_k$ the bridges of J in G. Let G' be the graph derived from G by deleting a bridge B_i.*

1. *$w(G, B_i) \geq 2$ for $1 \leq i \leq k$.*
2. *G' is nonseparable.* □

3 Existence of an Ambitus

Firstly we make following mild assumptions about the graph G:

Assumption 3.1 G is a nonseparable graph containing two distinguished vertices s and t and two internally vertex disjoint paths $P[s;t]$ and $Q[s;t]$. Further we assume that the bridges of $J = \{P\} \cup \{Q\}$ in G are B^P-, B^Q- or B^{PQ}-bridges, only. □

Definition 3.2 CARRIER OF A B^P- OR A B^Q-BRIDGE.

Let B be a B^P-bridge of $J = \{P\} \cup \{Q\}$ in G. Let $s_P(B)$ and $t_P(B)$ be its left- and rightmost vertices of attachment on P. Then we say $P[s_P(B); t_P(B)]$ is the *P-carrier* (or simply its *carrier*, when B is understood to be a B^P-bridge, from the context).

A Q-carrier of a B^Q-bridge is defined in an identical manner. □

Definition 3.3 THE COVERING RELATION.

A B^P-bridge B_1 *covers* another B^P-bridge B_2, if following two conditions are satisfied:

1. *The P-carrier of B_2, $P_2' = P[s_P(B_2); t_P(B_2)]$, is a subpath of the P-carrier of B_1, $P_1' = P[s_P(B_1); t_P(B_1)]$.*
2. *B_2 has a vertex of attachment on $P]s_P(B_1); t_P(B_1)[$.*

A B^P-bridge B_1 *covers* a vertex v, if $v \in V(P]s_P(B_1); t_P(B_1)[)$ and *maximally covers*, if, in addition, no B^P-bridge B_2 *covers* B_1.

A similar set of definitions holds for B^Q-bridges and the vertices of $Q]s;t[$. □

Definition 3.4 AN OFFENSIVE B^P-BRIDGE.

Let $v \in V(P]s;t[)$ be a vertex of attachment of a B^{PQ}-bridge of $J = \{P\} \cup \{Q\}$ in G. If a B^P-bridge, B, covers v then B is *offensive* with respect to v. □

Now, we are ready to prove the existence of an ambitus; the proof is constructive and readily provides an $O(|E| \cdot |V|)$ time algorithm.

Theorem 3.1 *Let G be an undirected graph with two distinguished vertices s and t such that it has two internally vertex-disjoint paths $P_0[s;t]$ and $Q_0[s;t]$. Then G also has two internally vertex-disjoint paths $P[s;t]$ and $Q[s;t]$ such that the cycle $J = \{P\} \cup \{Q\}$ is an ambitus.*

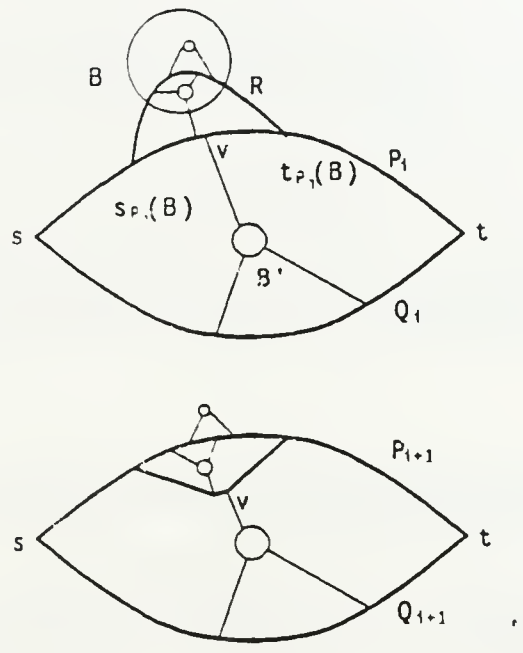


Figure 3: Modifying P.

PROOF.

Let $J_0 = \{P_0\} \cup \{Q_0\}$ be the cycle in the graph G. We describe a sequence of cycles, $J_0 = \{P_0\} \cup \{Q_0\}$, $J_1 = \{P_1\} \cup \{Q_1\}$, ..., $J_n = \{P_n\} \cup \{Q_n\}$, where J_n is an ambitus and $n \leq |V|$.

With each cycle we associate a number, v_i^{PQ}, the number of vertices of nuclei of B^{PQ}-bridges of J_i, *i.e.*,

$$v_i^{PQ} = \sum_{j=1}^{m_i} |V(N(B_j))| \leq |V(G)|,$$

where $B_1, B_2, \ldots, B_{m_i}$ are the B^{PQ}-bridges of J_i.

If J_i is an ambitus then J_{i+1} is undefined, and the sequence of cycles terminates, with $i = n$. Otherwise, we construct a cycle J_{i+1} as follows: Let B be an offensive B^P- or B^Q-bridge of J_i, covering a vertex v, a vertex of attachment of some B^{PQ}-bridge B' of J_i. Without loss of generality assume that B is a B^P-bridge. Let $s_{P_i}(B)$ and $t_{P_i}(B)$ be the left- and right-most vertices of attachment of B on P_i. Hence v lies on $P_i]s_{P_i}(B); t_{P_i}(B)[$, and $v \notin V(N(B_j))$, for all $1 \leq j \leq m_i$.

Let $R[s_{P_i}(B); t_{P_i}(B)]$ be a cross-cut of J_i between the vertices $s_{P_i}(B)$ and $t_{P_i}(B)$ such that R lies in the bridge B. Let

$$\begin{aligned} P_{i+1}[s;t] &= P_i[s; s_{P_i}(B)] * R[s_{P_i}(B); t_{P_i}(B)] * P_i[t_{P_i}(B); t], \\ Q_{i+1}[s;t] &= Q_i[s;t], \end{aligned}$$

and

$$J_{i+1} = \{P_{i+1}\} \cup \{Q_{i+1}\}.$$

Hence P_{i+1} and Q_{i+1} are internally vertex-disjoint.

Every vertex in the nuclei of B^{PQ}-bridges of J_i is also a vertex in the nuclei of B^{PQ}-bridges of J_{i+1}, and hence $v_i^{PQ} \leq v_{i+1}^{PQ}$. Furthermore, the vertex v belongs to the nucleus of a B^{PQ}-bridge of J_{i+1}, where as it used to be a vertex of J_i. Hence,

$$v_i^{PQ} < v_{i+1}^{PQ} \leq |V(G)|.$$

Since each modification strictly increments the v^{PQ}-number and since the v^{PQ}-number is bounded by $|V(G)|$ from above, this sequence of cycles must terminate with some $n \leq |V(G)|$, where J_n is an ambitus.

Since each modification step involves: (i) Finding an offensive B^P- or B^Q-bridge (ii) Finding a cross-cut R in the appropriate B^P- or B^Q-bridge, (iii) Modifying the paths P and Q, and (iv) Recomputing the bridges of the modified cycle, it takes no more than $O(|E|)$ time. Since such a step can be repeated at most $|V(G)|$-many times, the proof suggests an $O(|E| \cdot |V|)$ time algorithm to find an ambitus. □

4 Sketch of the Algorithm

For the sake of exposition, we sketch the algorithm FIND-AMBITUS without the implementation details. The algorithm, implemented in a straight-forward manner, does not produce a linear time algorithm. The implementation details, required to achieve the time complexity, are discussed in the two subsequent sections; and are based on the following two main ideas:

1. PATH-FINDING: The first idea involves a modification to the depth-first search and helps us to quickly find the required paths in a B^P-bridge.

2. AUXILIARY DATA STRUCTURE: The second idea involves an auxiliary data structure with the operation FIND-AND-UPDATE, which, in effect, determines the order in which B^P-bridges are to be examined.

4.1 The Algorithm

The algorithm consists of the following three mutually recursive algorithms: FIND-AMBITUS, ANALYZE-BRIDGES and ANALYZE-BRIDGE. The correctness of the algorithm is proven in the next subsection.

Algorithm FIND-AMBITUS(G, s, t):
begin

Let $\mathcal{B}^P$ be the set of B^P-bridges of J in G,and $\mathcal{V}^P$,
the set of vertices of attachment of the
B^{PQ}-bridges of J in G, on $P]s;t[$;
$P'[s;t] :=$ ANALYZE-BRIDGES$(P[s;t], \mathcal{B}^P, \mathcal{V}^P)$;

Let J' be $\{P'\} \cup \{Q\}$;

Let $\mathcal{B}^Q$ be the set of B^Q-bridges of J' in G, and $\mathcal{V}^Q$,
the set of vertices of attachment of the
B^{PQ}-bridges of J' in G, on $Q]s;t[$;
$Q'[s;t] :=$ ANALYZE-BRIDGES$(Q[s;t], \mathcal{B}^Q, \mathcal{V}^Q)$;

return $J'' = \{P'\} \cup \{Q'\}$;
end{**FIND-AMBITUS**} □

```
Algorithm ANALYZE-BRIDGES(L[s;t], B^L, V^L):
begin
    L'[s;t] := L[s;t];
    V' := V^L;
    Unmark bridges of B^L;

    MainLoop:
    until V' = ∅ loop

        Find-and-Update:
        Pair := ⟨v, B⟩
            (* v ∈ V' and B = an unmarked bridge, maximally covering v,
             * if such a bridge exists; B = ⊥, otherwise. *)
        V' := V' \ {v};

        if B ≠ ⊥ then
            Let B' ⊆ B^L be the set of bridges, whose carriers intersect
                with the subpath L]s_L(B);t_L(B)[;
            for every B' ∈ B' loop
                (* Let W(B') = vertices of attachment of B'. *)
                if B' is unmarked then
                    if B' ≠ B cand
                        B' has a vertex of attachment on L]s_L(B);t_L(B)[
                    then
                        V' := V' ∪ W(B');
                    end{if }
                    Mark B';
                end{if };
            end{loop };
        end{if };
        end{Find-and-Update};

        if B ≠ ⊥ then
            Let L'' = L[s_L(B);t_L(B)] be the carrier of B;
            Let G_B = ⟨E(B) ∪ E(L'')⟩;
            R[s_L(B);t_L(B)] := ANALYZE-BRIDGE(G_B, s_L(B), t_L(B), L'');
            L'[s;t] := L'[s;s_L(B)] * R[s_L(B);t_L(B)] * L'[t_L(B);t];
        end{if };
    end{MainLoop};

    return L'[s;t];
end{ANALYZE-BRIDGES}  □
```

Algorithm ANALYZE-BRIDGE$(G_B, s, t, Q[s;t])$:
begin
(* G_B is composed of the bridge, B, and its carrier, $Q[s;t]$ *)

Find a path $P[s;t]$ from s to t in B;
(* $P[s;t]$ and $Q[s;t]$ are two internally vertex disjoint paths
* in G_B and $J = \{P\} \cup \{Q\}$ is a cycle in G_B. *)

Let $\mathcal{B}^P$ be the set of B^P-bridges of J in G_B, and $\mathcal{V}^P$,
the set of vertices of attachment of the B^{PQ}-bridges of J
in G_B, on $P]s;t[$;

$R[s;t] :=$ ANALYZE-BRIDGES$(P[s;t], \mathcal{B}^P, \mathcal{V}^P)$;

return $R[s;t]$;
end{**ANALYZE-BRIDGE**} □

4.2 The Correctness of the Algorithm Find-Ambitus

For the proof of correctness of the algorithms, we need to define the following notations:

- A sequence of paths: $P_0[s;t]$, $P_1[s;t]$, ..., $P_i[s;t]$, ..., where $P_0[s;t] = P[s;t]$ and $P_i[s;t]$ is the modified path obtained at the end of the i^{th} iteration of the MAINLOOP of the algorithm ANALYZE-BRIDGES.
- A sequence of sets of vertices: $\mathcal{V}_0$, $\mathcal{V}_1$, ..., $\mathcal{V}_i$, ..., where $\mathcal{V}_0 = \mathcal{V}^P$ and $\mathcal{V}_i$ is the modified $\mathcal{V}'$ at the end of the i^{th} iteration of the MAINLOOP of the algorithm ANALYZE-BRIDGES.

Definition 4.1 A path $P_i[s;t]$ is *well-defined* with respect to the cycle $J = \{P\} \cup \{Q\}$, if

1. *It is simple and internally vertex-disjoint with* $Q[s;t]$.
2. *There is a sequence of vertices:* $v_1, v'_1, v_2, v'_2, \ldots, v_p, v'_p$ *on* $P]s;t[$ *arranged in a left-to right order such that* $P_i[s;t]$ *can be written as the concatenation of a sequence of alternating common-sections and cross-cuts as follows:*

$$P[s;v_1] * R[v_1;v'_1] * \cdots * R[v_p;v'_p] * P[v'_p;t].$$

The vertices $v_1, v'_1, v_2, v'_2, \ldots, v_p, v'_p$ dissect the path P_i into internally vertex disjoint subpaths $P_i[s;v_1] = P[s;v_1]$, $P_i[v_1;v'_1] = R[v_1;v'_1]$, ..., $P_i[v_p;v'_p] = R[v_p;v'_p]$ and $P_i[v'_p;t] = P[v'_p;t]$. □

Let

$$\mathcal{W}_i = V(P) \cap V(P_i) = V(P[s;v_1]) \cup V(P[v'_1;v_2]) \cup \cdots \cup V(P[v'_p;t]),$$

and

$$\overline{\mathcal{W}_i} = V(P) \setminus V(P_i) = V(P]v_1;v'_1[) \cup V(P]v_2;v'_2[) \cup \cdots \cup V(P]v_p;v'_p[).$$

Let P_i be well-defined, with the sequence of vertices $v_1, v'_1, v_2, v'_2, \ldots, v_p, v'_p$ dissecting it into internally vertex disjoint subpaths as in the Definition 4.1. Thus $J_i = \{P_i\} \cup \{Q_i\}$ is a cycle in

G. Let $\mathcal{B}_i$, $\mathcal{N}_i$ and $\mathcal{A}_i$ stand, respectively, for the set of B^{PQ}-bridges of J_i in G, the vertices of their nuclei and their vertices of attachment on $P_i]s;t[$. Also, we say that the B^P-bridges of J_i are *partitioned*, if every B^P-bridge of J_i in G has all its vertices of attachment on one of the subpaths $P_i[s;v_1]$, $P_i[v_1;v_1']$, $P_i[v_1';v_2]$, ..., $P_i[v_p;v_p']$ or $P_i[v_p';t]$.

Lemma 4.1 (Main Technical Lemma) *For all $i \geq 0$,*

1. *$P_i[s;t]$ is well-defined with respect to J.*
2. *(a) $\overline{\mathcal{W}_i} \subseteq \mathcal{N}_i$.*
   (b) $\mathcal{V}_i \cap \mathcal{W}_i \subseteq \mathcal{A}_i$.
3. *(a) The B^P-bridges of J_i in G are partitioned.*
   (b) If there is an offensive B^P-bridge B' of J_i in G, then B' is also a B^P-bridge of J. Furthermore, B' is unmarked and covers a vertex $w \in \mathcal{V}_i \cap \mathcal{W}_i$ of J.

PROOF.

The proof is by a *double induction* on i and the size of the graph.
BASIS STEP: (1) follows from the fact that $P_0[s;t] = P[s;t]$. (2a) and (b) follow from the facts that $\mathcal{W}_0 = V(P[s;t])$, $\overline{\mathcal{W}_0} = \emptyset$ and $\mathcal{V}_0 \cap \mathcal{W}_0 = \mathcal{A}_0$. (3a) and (b) holds trivially, since every B^P-bridge of J_0 is an unmarked B^P-bridge of J, and if offensive, it covers a vertex $w \in \mathcal{A}_0 = \mathcal{V}_0 \cap \mathcal{W}_0$.
INDUCTION STEP: Let $\langle v, B\rangle$ be the *Pair* chosen in the i^{th} iteration of MAINLOOP. Hence $v \in \mathcal{V}_{i-1}$.

(CASE1). *There is no unmarked bridge $B \in \mathcal{B}^P$, covering v.* Hence $B \neq \perp$.
(1) follows from the fact that $P_i[s;t] = P_{i-1}[s;t]$. (2a) holds, since $\overline{\mathcal{W}_i} = \overline{\mathcal{W}_{i-1}} \subseteq \mathcal{N}_{i-1} = \mathcal{N}_i$. (2b) holds, since $\mathcal{V}_i \cap \mathcal{W}_i = (\mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}) \setminus \{v\} \subseteq (\mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}) \subseteq \mathcal{A}_{i-1} = \mathcal{A}_i$. (3a) is trivially true, since $P_i = P_{i-1}$ and $J_i = J_{i-1}$.
(3b) Let B' be an offensive B^P-bridge of J_i in G. Since $J_i = J_{i-1}$, B' is also an offensive B^P-bridge of J_{i-1}. By the inductive hypothesis, at the end of the $(i-1)^{\text{th}}$ iteration, B' is an unmarked B^P-bridge of J and covers a vertex $w \in \mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}$. If w is distinct from v then $w \in \mathcal{V}_i \cap \mathcal{W}_i$; and if $w = v$ then $v \in \mathcal{W}_{i-1}$ and there is an unmarked bridge $B' \in \mathcal{B}^P$, covering v, which is false by assumption.

(CASE2). *$B \in \mathcal{B}^P$ is an unmarked bridge, maximally covering v.* Since every bridge covering $w \in \overline{\mathcal{W}_{i-1}}$ is marked, $v \in \mathcal{W}_{i-1}$.

Since $P_{i-1}[s;t]$ is well-defined (inductive hypothesis) it can be written as:

$$P[v_0'(=s);v_1] * R[v_1;v_1'] * \ldots * R[v_p;v_p'] * P[v_p';v_{p+1}(=t)].$$

(1) Since B is unmarked, $P[s_P(B);t_P(B)]$ is a subpath of $P[v_k';v_{k+1}]$, for some $0 \leq k \leq p$. Hence all the vertices of attachment of B lie on the subpath $P[v_k';v_{k+1}]$ and B is also a B^P-bridge of J_{i-1}.

Since $R[s_P(B);t_P(B)] \in G_B$ and by the inductive hypothesis $R[s_P(B);t_P(B)]$ is a simple path internally vertex disjoint with $P[s_P(B);t_P(B)]$,

$$\begin{aligned} P_i[s;t] &= P_{i-1}[s;s_P(B)] * R[s_P(B);t_P(B)] * P_{i-1}[t_P(B);t] \\ &= P[s;v_1] * \cdots P[v_k';s_P(B)] * R[s_P(B);t_P(B)] * P[t_P(B);v_{k+1}] * \cdots * P[v_p';t]. \end{aligned}$$

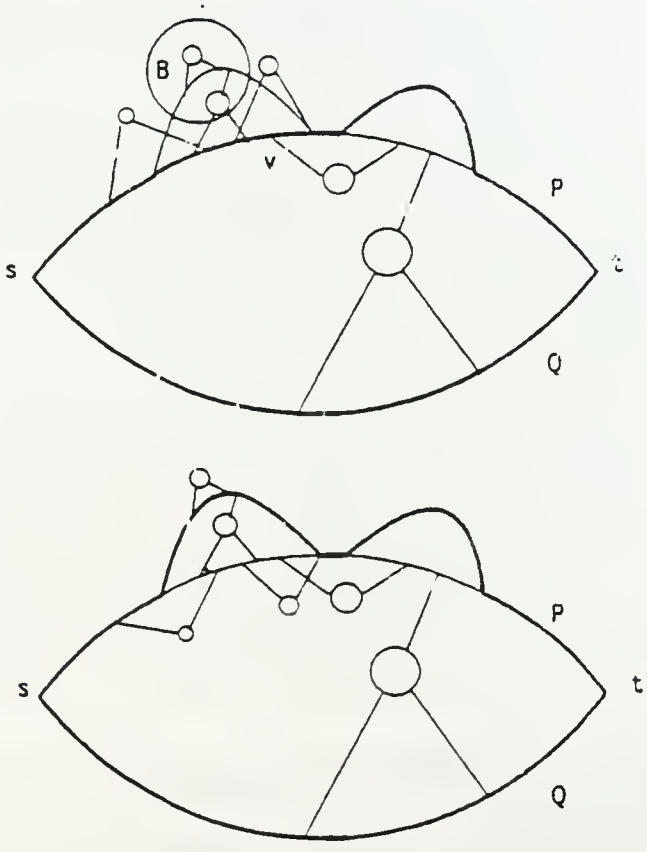


Figure 4:

is well-defined. $\mathcal{W}_i = \mathcal{W}_{i-1} \setminus V(P]s_P(B); t_P(B)[)$.

(2a) Since $v \in \mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}$, $v \in \mathcal{A}_{i-1}$. Hence there is a cross-cut $Y[v; z]$ of J_{i-1}, where $z \in Q]s; t[$. Hence $Y[v; z]$ meets $P[s_P(B); t_P(B)]$ only in v and otherwise avoids it. Since $R[s_P(B); t_P(B)]$ avoids $P_{i-1}[s; t]$ except for its end vertices, $\{P[s_P(B); v], P[v; t_P(B)], Y[v; z]\}$ are the arms of a Y-graph with the center v. Hence $V(P]s_P(B); t_P(B)[) \subseteq \mathcal{N}_i$. Using the inductive hypothesis, we see that $\overline{\mathcal{W}_i} \subseteq \mathcal{N}_i$.

(2b) For all $x \in \mathcal{W}_i$, we show that $x \in \mathcal{V}_i \Longrightarrow x \in \mathcal{A}_i$.

Assume that $x \in \mathcal{V}_i \cap \mathcal{V}_{i-1}$. In this case, $x \in \mathcal{V}_{i-1} \cap \mathcal{W}_{i-1}$ and is a vertex of attachment of a bridge of $\mathcal{B}_{i-1}$ on $P_{i-1}]s; t[\setminus P]s_P(B); t_P(B)[$. Hence there is a cross-cut, $L[x; y]$, between x and a vertex $y \in Q]s; t[$ in J_{i-1}. Since $R[s_P(B); t_P(B)]$ is in B, $L[x; y]$ meets J_i only in x and y and otherwise avoids it. Hence $x \in \mathcal{A}_i$.

Otherwise, $x \in \mathcal{V}_i \setminus \mathcal{V}_{i-1}$. Hence x must be a vertex of attachment of a B^P-bridge, B', of J in G, where B' has a vertex of attachment, y on $P]s_P(B); t_P(B)[$. Hence there is a cross-cut, $L[x; y]$

between x and y in J. $L[x;y]$ avoids every B^P-bridge of J and hence the subpaths $R[v_1;v'_1]$, ..., $R[v_k;v'_k]$, $R[s_P(B);t_P(B)]$, $R[v_{k+1};v'_{k+1}]$, ..., $R[v_p;v'_p]$. Since $x \in \mathcal{W}_i$, x is a vertex on P_i. As a result, $L[x;y]$ is a simple path of G that meets J_i in x and otherwise avoids it. But by (2a), $y \in \overline{\mathcal{W}_i} \subseteq \mathcal{N}_i$ and hence $x \in \mathcal{A}_i$. $\mathcal{V}_i \cap \mathcal{W}_i \subseteq \mathcal{A}_i$.

(3a) Let B' be a B^P-bridge of J_i in G. If B' has a vertex of attachment, x, on $R]s_P(B);t_P(B)[$ then all its vertices of attachment lie on $R[s_P(B);t_P(B)]$. Suppose not. Then there is a cross-cut $L[x;y]$ of J_i between x and some vertex y, where $y \in V(P_i[s;t]) \setminus V(R[s_P(B);t_P(B)])$. Since all the vertices of $P]s_P(B);t_P(B)[$ belong to $\mathcal{N}_i$, $L[x;y]$ avoids $P[s_P(B);t_P(B)]$. But this implies that there is a path from $x \in N(B)$ to a vertex $y \in G \setminus B$ that avoids the vertices of attachment of B on J in G.

Hence, B' is a B^P-bridge of J_i in G such that it has no vertex of attachment on $R]s_P(B);t_P(B)[$, and thus a B^P-bridge of J_{i-1}. Notice that if B' has a vertex of attachment on $P[s;s_P(B)]$ and one on $P[t_P(B);t]$ then B' is an offensive B^P-bridge of J_{i-1}. Hence B' is a B^P-bridge of J and is unmarked at the end of the $(i-1)^{\text{th}}$ iteration and the carrier of B is a subpath of the carrier of B'. Since B, by assumption, is a maximal bridge covering v, B has no vertex of attachment on $P]s_P(B');t_P(B')[$. Hence $s_P(B') = s_P(B)$, $t_P(B') = t_P(B)$; and all the vertices of attachment of B' lie on $R[s_P(B');t_P(B')]$.

Assume that all the vertices of attachment of B' lie on $P[s;s_P(B)]$ or on $P[t_P(B);t]$ (say the former). But by the inductive hypothesis, all the vertices of attachment of B' lie on some subpath of P_{i-1}, $P_{i-1}[v'_j;v_{j+1}]$ $(0 \le j \le k-1)$, on $P_{i-1}[v_j;v'_j]$ $(1 \le j \le k)$, or on $P_{i-1}[v'_k;s_P(B)]$. Hence B' has all its vertices of attachment on $P_i[v'_j;v_{j+1}]$ $(0 \le j \le k-1)$, on $P_i[v_j;v'_j]$ $(1 \le j \le k)$, or on $P_i[v'_k;s_P(B)]$.

(3b) Let B' be an offensive B^P-bridge of J_i in G. Let $s_{P_i}(B')$ and $t_{P_i}(B')$ be the left- and right-most vertices of attachment of B' on P_i. then there is an edge $e = [x,y]$ included in $\mathcal{B}_i$ such that x is on $P_i]s_{P_i}(B');t_{P_i}(B')[$.

(*i*) B' has all its vertices of attachment on $P_i[s;t] \setminus P_i]v'_k;v_{k+1}[$. Since B^P-bridges of J_i in G are partitioned (by (3a)), all the vertices of attachment of B' lie on some subpath $P_i[a;b]$, where $P_i[a;b]$ is one of $P_i[v_j;v'_j]$ $(1 \le j \le p)$ and $P_i[v'_j;v_{j+1}]$ $(0 \le j \le p$ and $j \ne k)$. Hence B' is a B^P-bridge of J_{i-1}, with all its vertices of attachment on $P_{i-1}[a;b]$. If B' is not an offensive B^P-bridge of J_{i-1} then the edge e is included in some B^P-bridge, B'', of J_{i-1}. Since B'' has a vertex of attachment at $x \in P_{i-1}]a;b[$, all its vertices of attachment lie on $P_{i-1}[a;b]$ and hence B'' is a B^P-bridge of J_i containing the edge e, thus refuting the assumption. Hence B' is an offensive B^P-bridge of J_{i-1} and the rest follows from the inductive hypothesis.

(*ii*) B' has all its vertices of attachment on $P_i[v'_k;s_P(B)]$ or $P_i[t_P(B);v_{k+1}]$, say the former. Hence B' is a B^P-bridge of J_{i-1}, with all the vertices of attachment on $P_{i-1}[v'_k;s_P(B)]$. If B' is not an offensive B^P-bridge of J_{i-1} then e is included in some B^P-bridge, B'', of J_{i-1}. Since B'' has a vertex of attachment x on $P_{i-1}]v'_k;s_P(B)[$ all its vertices of attachment lie on $P_{i-1}[v'_k;v_{k+1}]$. Since $P_{i-1}[v'_k;v_{k+1}] = P[v'_k;v_{k+1}]$ both B' and B'' are B^P-bridges of J in G, and unmarked at the end of the $(i-1)^{\text{th}}$ iteration. If B'' has no vertex of attachment on $P]s_P(B);t_P(B)[$ then B'' is a B^P-bridge of J_i, thus refuting the assumption; and if it has a vertex of attachment on $P]s_P(B);t_P(B)[$ then $x \in \mathcal{V}_i \cap \mathcal{W}_i$. If, on the other hand, B' is an offensive B^P-bridge of J_{i-1}, then (3b) follows, as in (*i*), from the inductive hypothesis.

(*iii*) B' has all its vertices of attachment on $P_i[s_P(B);t_P(B)]$. Hence x lies on $P_i]s_P(B);t_P(B)[$. Hence there is a cross-cut $L[x;z]$ of J_i, between x and a vertex z on $Q]s;t[$,

where the cross-cut $L[x;z]$ includes the edge e. Since $x \in N(B)$ and $z \in G \setminus B$, $L[x;z]$ must meet $P]s_P(B);t_P(B)[$, at some vertex, and an appropriate subpath of $L[x;z]$, $L[x;z']$ meets $R]s_P(B);t_P(B)[$ only in x, $P]s_P(B);t_P(B)[$ only in z', and otherwise avoids $R[s_P(B);t_P(B)]$, and $P[s_P(B);t_P(B)]$. Furthermore $L[x;z']$ is a path in G_B. Hence B is a B^P-bridge of J with at least one vertex of attachment on $P]s_P(B);t_P(B)[$.

If B' has no vertex of attachment on $R]s_P(B);t_P(B)[$, then B' is an unmarked B^P-bridge of J in G; and the carrier of B is a subpath of the carrier of B'. Since B has a vertex of attachment on $P]s_P(B);t_P(B)[$, it also has a vertex of attachment on $P]s_P(B');t_P(B')[$. But, this is impossible, since B is a maximal unmarked bridge, by assumption.

Hence B' has a vertex of attachment on $R]s_P(B);t_P(B)[$. Let $R[s_P(B);t_P(B)] = P''$, $P[s_P(B);t_P(B)] = Q''$ and $J'' = \{P''\} \cup \{Q''\}$ in G_B. Then B' is an offensive B^P-bridge of J'' in G_B. But by inductive hypothesis, this is impossible. □

Corollary 4.2 *Let G be as in the assumption 3.1. Let $\mathcal{B}^P$ be the set of its B^P-bridges and $\mathcal{V}^P$, the set of vertices of attachment of its B^{PQ}-bridges on $P]s;t[$. Let $P'[s;t]$ be the path returned by the algorithm* ANALYZE-BRIDGES$(P[s;t], \mathcal{B}^P, \mathcal{V}^P)$ *and $J' = \{P'\} \cup \{Q\}$.*

1. *$P'[s;t]$ is a simple path internally vertex disjoint with $Q[s;t]$.*
2. *The number of offensive B^P-bridges of $J' = 0$.*
3. *The number of offensive B^Q-bridges of J' = the number of offensive B^Q-bridges of J.*

PROOF.

When the algorithm ANALYZE-BRIDGES terminates (after n iterations of the MAINLOOP)

$$P'[s;t] = P_n[s;t] \qquad \text{and} \qquad \mathcal{V}_n = \mathcal{V}_n \cap \mathcal{W}_n = \emptyset.$$

(1) and (2) follow from (1) and (3b) of the previous Lemma.

(3) Since $P'[s;t]$ and $P[s;t]$ are internally vertex disjoint with $Q[s;t]$, every B^Q-bridge of J' is a B^Q-bridge of J and *vice versa*.

Let B be an offensive B^Q-bridge of J'. Then there is an edge $e = [x,y]$ included in a B^{PQ}-bridge of J' such that x is on $Q]s_P(B);t_P(B)[$. If B is not an offensive B^Q-bridge of J then e is included in some B^Q-bridge, B', of J with a vertex of attachment at x. But since B' is also a B^Q-bridge of J', this is a contradiction. The converse holds by a similar argument. □

Corollary 4.3 *The paths $P'[s;t]$ and $Q'[s;t]$, returned by the algorithm* FIND-AMBITUS, *are simple and internally vertex disjoint, and $J'' = \{P'\} \cup \{Q'\}$ is an ambitus.* □

5 Implementation: Path-Finding

In this section, we present an algorithm to obtain a representation of the graph G that allows efficient implementation of many of the basic steps of the algorithm. This section is essentially Hopcroft and Tarjan's Algorithm (1973)[3] for planarity-testing, with appropriate modifications. The representation consists of a palmtree $\mathcal{P}$ of the graph G, with its tree edges and back edges appropriately directed together with a partition of the edges of the graph into a set of internally vertex disjoint simple paths. We also describe how this representation will help us to recursively determine the paths in the graphs G or G_B together with the sets $\mathcal{B}^P$ and $\mathcal{V}^P$; this representation is also used in finding the vertices of attachment of a B^P-bridge of G or G_B.

5.1 Building the Palm-Tree

Let $G = (E, V)$ be a nonseparable graph, with two distinguished vertices s and t, where G satisfies the Assumption 3.1. Let

$$P[s;t] = v_p(=s), v_{p-1}, \ldots, v_1, v_0(=t),$$

and

$$Q[s;t] = w_0(=s), w_1, \ldots, w_{q-1}, w_q(=t).$$

be two internally vertex-disjoint paths in G that meet each other only in their end vertices.

Let $\widehat{\Phi}: E(G) \to \mathbf{N} \times \mathbf{N}$ be a function defined on the edges $[u, v]$ of G as follows:

$$\widehat{\Phi}([u,v]) = \begin{cases} \langle 0, 0\rangle, & \text{if } [u,v] = [w_0, w_1]; \\ \langle 0, |V|+1\rangle, & \text{if } [u,v] \in E(J) \setminus \{[w_0, w_1]\}; \\ \langle |V|+1, |V|+1\rangle, & \text{if } [u,v] \in E(G) \setminus E(J). \end{cases}$$

Assume that the graph G is represented by adjacency lists $A(v)$, ordered according to increasing values of $\widehat{\Phi}$, under a lexico-graphic ordering. Since $\widehat{\Phi}([u,v])$, for each edge in G, can be calculated in $O(|E|)$ time, and the adjacency lists can be ordered in $O(|E|)$ time, using a stable radix sort twice, such a representation can be obtained in $O(|E|)$ time. This can be implemented as follows:

```
begin
      for i := 0 to |V| + 1 loop  BUCKET(i) := ∅;

      for every [v, w] ∈ E(G) loop
            SECOND := second(Φ̂([v, w]));
            Add [v, w] to end of BUCKET(SECOND);
      end{loop };

      E' := ∅;
      for i := 0 to |V| + 1 loop
            for [v, w] ∈ BUCKET(i) loop Add [v, w] to end of E';
            end{loop };
            BUCKET(i) := ∅;
      end{loop };

      for [v, w] ∈ E' loop
            FIRST := first(Φ̂([v, w]));
            Add [v, w] to end of BUCKET(FIRST);
      end{loop };

      for every v ∈ V(G) loop A(v) := ∅;
      for i := 0 to |V| + 1 loop
            for [v, w] ∈ BUCKET(i) loop Add w to the end of A(v);
      end{loop };
end.  □
```

Next, we systematically explore the graph using a depth-first search starting at s. The DFS generates a palmtree $\mathcal{P}$, and numbers the vertices by their DFS-number in the range $(1..|V|)$. *Henceforth, we identify the vertices by their* DFS-*number.*

If v is a vertex, then let S_v stand for the set of vertices u reachable from a descendant of v by a single back edge. Let

$$\text{LOW1}(v) = \text{MIN}(\{v\} \cup S_v),$$

and

$$\text{LOW2}(v) = \text{MIN}(\{v\} \cup (S_v - \{\text{LOW1}(v)\})).$$

LOW1(v) is the (*first*) *lowest* vertex below v reachable from a descendant of v by a single back edge, and LOW2(v) is the *second lowest* vertex below v reachable from a descendant of v by a single back edge. By convention, LOW values of v are equal to v, if they are not defined.

The LOW values of a vertex v depend only on the LOW values of children of v and on the back edges leaving v—thus it is easy to calculate LOW values by appropriately modifying DFS algorithm. (see Path-finder algorithm in Hopcroft and Tarjan (1973)[3] and (1974)[4]).

We make the following observations:

1. The ordering induced by $\widehat{\Phi}$ ensures that the DFS visits the vertices of P and Q before it enters any of the bridges of J, and that the edge $[v_{p-1}, v_p]$ is a back-edge and all other edges of J are tree-edges.

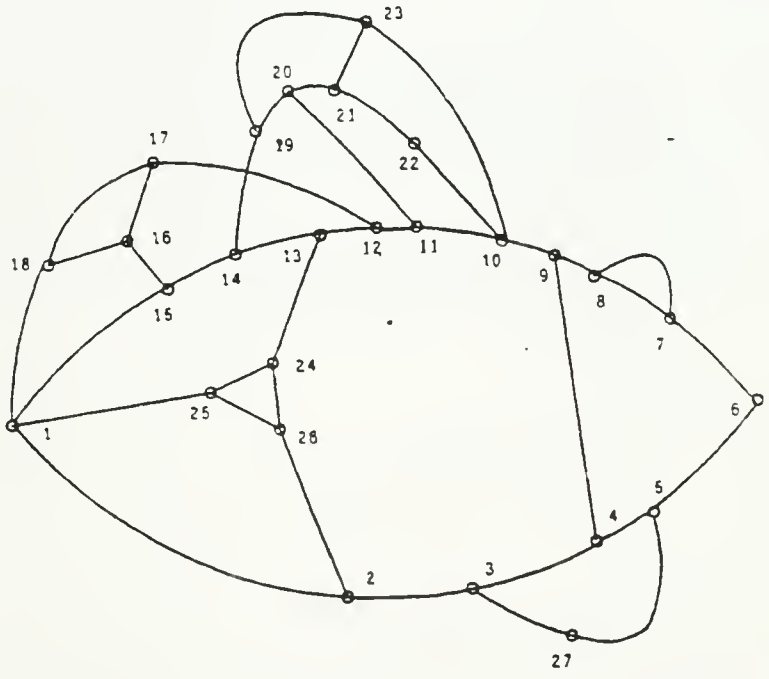


Figure 5: The Graph G.

2. After the DFS each of the edges is directed; the tree edges are directed from a smaller vertex to a larger vertex and the back edges from a larger vertex to a smaller vertex. Thus each edge appears once in the adjacency lists of $\mathcal{P}$. If $u \rightarrow v$ is an edge of $\mathcal{P}$ then by convention, $\widehat{\Phi}(u \rightarrow v) = \widehat{\Phi}([u, v])$.

In the next step, we use a *second* depth-first search, carried out in a special order, to divide the graph into a set of simple paths which may be assembled in order to build the ambitus. To generate paths, we sort the adjacency lists of $\mathcal{P}$ according to the LOW values. For this purpose, we define a function $\Phi: E(G) \rightarrow \mathbf{N} \times \mathbf{N}$, on the edges $u \rightarrow v$ of $\mathcal{P}$ as follows:

$$\Phi'(u \rightarrow v) = \begin{cases} \langle v, u \rangle, & \text{if } u \rightarrow v \text{ is a back edge;} \\ \langle \text{LOW1}(v), u \rangle, & \text{if } u \rightarrow v \text{ is a tree edge and} \\ & \quad \text{LOW2}(v) \geq u; \\ \langle \text{LOW1}(v), \text{LOW2}(v) \rangle, & \text{if } u \rightarrow v \text{ is a tree edge and} \\ & \quad \text{LOW2}(v) < u. \end{cases}$$

and

$$\Phi(u \rightarrow v) = \text{MIN}\left(\Phi'(u \rightarrow v), \widehat{\Phi}(u \rightarrow v)\right),$$

where MIN is taken with respect to the lexicographic ordering.

We calculate $\Phi(u \rightarrow v)$ for each edge $u \rightarrow v$ of $\mathcal{P}$ and order the adjacency lists according to increasing values of Φ (under a lexicographic ordering), using a stable radix sort twice to

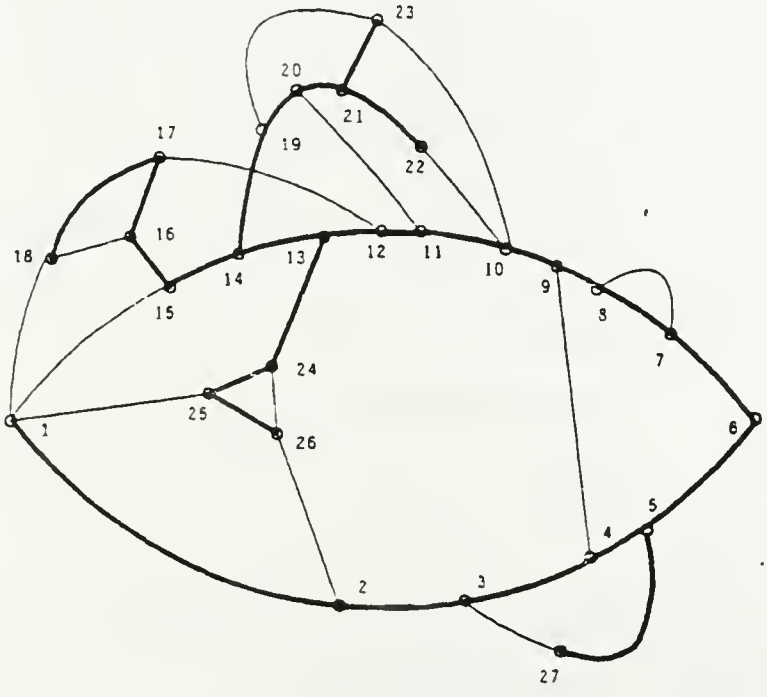


Figure 6: The palm tree $\mathcal{P}$ of G.

achieve an $O(|E| + |V|)$ time bound. The algorithm is similar to the one described earlier with $\hat{\Phi}$ replaced by Φ.

Now we generate paths by applying depth-first search to $\mathcal{P}$, using the new adjacency lists. Each time we traverse a tree edge we add it to the path being built. Each time we traverse a back edge, the back edge becomes the last edge of the current path. The next stage starts a new path. Thus each path consists of a sequence of tree edges followed by a back edge. Since each path is completely built before the next stage with a new path is started, the order in which the paths are generated induces a linear order among the paths. Furthermore, the edges $E(G)$ of the graph, G, are partitioned into a set of internally vertex disjoint simple paths; this partition of $E(G)$ is represented by associating with each edge $u \rightarrow v$ of $\mathcal{P}$, PATH$(u \rightarrow v)$, the unique path containing the edge. To accomplish this, we use the following steps:

```
Algorithm PATH-FINDER(u):
begin
    for v ∈ A(u) loop
        if u → v is a tree edge then
            L := L * [u, v];
            PATH(u → v) := L;
            PATH-FINDER(v);
        elsif u → v is a back edge then
            L := L * [u, v];
            PATH(u → v) := L;
            (* This completes the generation of the path L *)
            L := null path;
        end{if };
    end{loop };
end{PATH-FINDER}  □

Algorithm MAIN-PATH-FINDER(P):
begin
    L := null path;
    PATH-FINDER(v);
end{MAIN-PATH-FINDER}  □
```

The algorithm PATH-FINDER requires $O(|E|)$ time to find the paths in the graph $G = (V, E)$; the total number of edges added to paths is $O(|V| + |E|)$, and PATH-FINDER is just a depth-first search with a few additional operations to construct paths.

Lemma 5.1 *Let L be the first path found by the algorithm* PATH-FINDER. *Then $L = J = \{P\} \cup \{Q\}$.*

PROOF.

Since $\Phi([s, w_1]) = \langle 0, 0\rangle$, $\Phi([s, w_1]) < \Phi([u', v'])$, for every edge $[u', v'] \in E(G) \setminus \{[w_0, w_1]\}$, and $s \to w_1$ is the first tree edge of L. If $[u, v]$ is an edge of J, other than $[s, w_1]$, then $\Phi([u, v]) < \Phi([u', v'])$ for every edge $[u', v'] \in E(G) \setminus E(J)$, and hence $L = J = s(= w_0) \to w_1 \to \cdots \to w_{q-1} \to w_q(= v_0 = t) \to v_1 \to \cdots \to v_{p-1} \to s(= v_p)$, where all but the last edge $v_{p-1} \to v_p$ are tree edges. □

Lemma 5.2 *Let $L[f; l]$ be a path found by the algorithm* PATH-FINDER. *If L is not the initial path then*

1. *L is a simple path.*
2. *L contains exactly two vertices (f and l) in common with previously generated paths.*
3. *If we consider the back edges not yet used when the first edge of L is traversed, then l is the lowest vertex reachable via such a back edge from any descendant of f.*
4. *If $v \in L]f; l[$ then l is the lowest vertex reachable from any descendant of v via any back edge.*

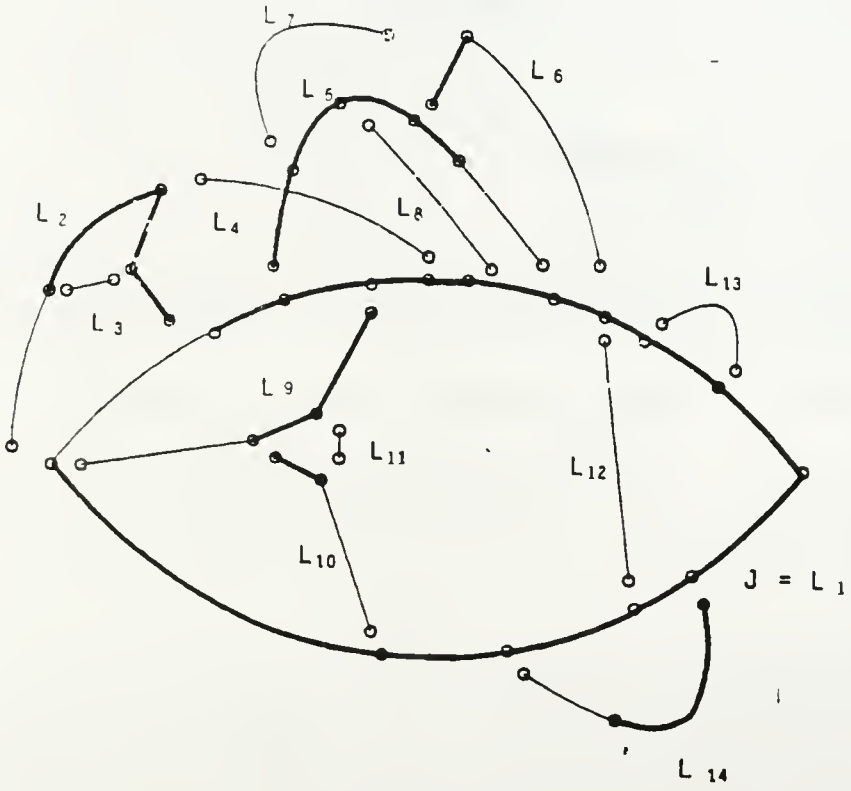


Figure 7: The paths in $\mathcal{P}$.

PROOF.

(3) and (4) are immediate consequence of the ordering of the adjacency lists. (1) and (2) follow from (3) and (4), the properties of DFS algorithm and the fact that $\mathcal{P}$ is nonseparable. (See Hopcroft and Tarjan (1973)[3]) □

Definition 5.1 SEGMENTS.

Let J be the cycle in G. When J is removed, G falls into several connected pieces, called *segments* of J in G. Each segment, S, consists either of a single back edge $e = u \rightarrow v$, or of a tree edge $e = u \rightarrow v$ plus a subtree with root at v plus all back edges leading from the subtree—here, $u \in V(J)$. We say that the segment, S, is associated with the edge e.

Let S' be a segment associated with an edge $e' = u' \rightarrow v'$. Let $L = \text{PATH}(u' \rightarrow v')$. If S' consists of the single back edge e' then the set of *segments* of L in S' is empty. Otherwise, S' consists of the tree edge e' plus a subtree with root v' plus all back edges leading from the

subtree. When L is removed, S' falls into several connected pieces, called the *segments* of L in S'. Each segment, S, consists either of a single back edge $e = u \rightarrow v$, or of a tree edge $e = u \rightarrow v$ plus a subtree with root at v plus all back edges leading from the subtree—here, $u \in V(L)$. We say that the segment. S, is associated with the edge e.

Note that the definition of a segment is recursive, and is based on the structure of the palm-tree. □

5.2 Θ-Graph and its Bridges.

The following definition will be useful in the next subsection.

Definition 5.2 Let $J = \{P[s;t] \cup Q[s;t]\}$ be a cycle in G. Let $R[x;y]$ be cross-cut of J between x, an internal vertex of Q, and y, an internal vertex of P. The subgraph, $\Theta = \{J\} \cup \{R\}$, of G, is called a Θ-*Graph* in G. A bridge B of Θ in G is a

1. B^P-BRIDGE, *if* B *has* at least one *vertex of attachment on* $P]s;t[$, *but* none *on* $Q]s;t[$ or $R]x;y[$.
2. B^Q-BRIDGE, *if* B *has* at least one *vertex of attachment on* $Q]s;t[$, *but* none *on* $P]s;t[$ *or* $R]x;y[$.
3. B^R-BRIDGE, *if* B *has* at least one *vertex of attachment on* $R]x;y[$, *but* none *on* $P]s;t[$ *or* $Q]s;t[$.
4. B^{PQR}-BRIDGE, *if* B *has* at least one *vertex of attachment on each of the two or more out of the three subpaths:* $P]s;t[$, $Q]s;t[$ *and* $R]x;y[$.

Notice that the B^P-bridges of Θ are B^P-bridges of J, and if $\mathcal{V}^P_\Theta$ = the set of vertices of attachment of the B^{PQR}-bridges of Θ on $P]s;y[$ and $P]y;t[$ then $\mathcal{V}^P = \mathcal{V}^P_\Theta \cup \{y\}$ = the set of vertices of attachment of the B^{PQ}-bridges of J on $P]s;t[$. □

5.3 Using the Palm-Tree

Next we describe how the palm tree can be used to find paths and bridges of the graphs quickly, in a recursive manner. Also, we classify B^P-bridges into: (*i*) *Normal* B^P*-bridges* and (*ii*) *Special* B^P*-bridges*, and use this classification in the recursive application of the algorithm, since the bridges of different classes have to be treated somewhat differently.

(1) The Basis Step.

By Lemma 5.1, the first path found by the PATH-FINDER is PATH$(s \rightarrow w_1) = J$, the cycle consisting of the tree edges $s \rightarrow w_1 \rightarrow \cdots \rightarrow w_{q-1} \rightarrow t \rightarrow v_1 \rightarrow \cdots \rightarrow v_{p-1}$ and the back edge $v_{p-1} \rightarrow s$, where

$$s < w_1 < \cdots < w_{q-1} < t < v_1 < \cdots < v_{p-1}.$$

Notice that every bridge of J in G is a segment of J in G. Let u be a vertex on $P]s;t[$, and $e = u \rightarrow v$ be an edge, not belonging to P. Let B be the bridge of J in G, equal to the segment associated with e.

If e is a back edge and $s < v < t$ then B is a degenerate B^{PQ}-bridge. If e is a tree edge and $s <$ LOW1$(v) < t$, or $s <$ LOW2$(v) < t$ then B is a proper B^{PQ}-bridge. Otherwise, B is

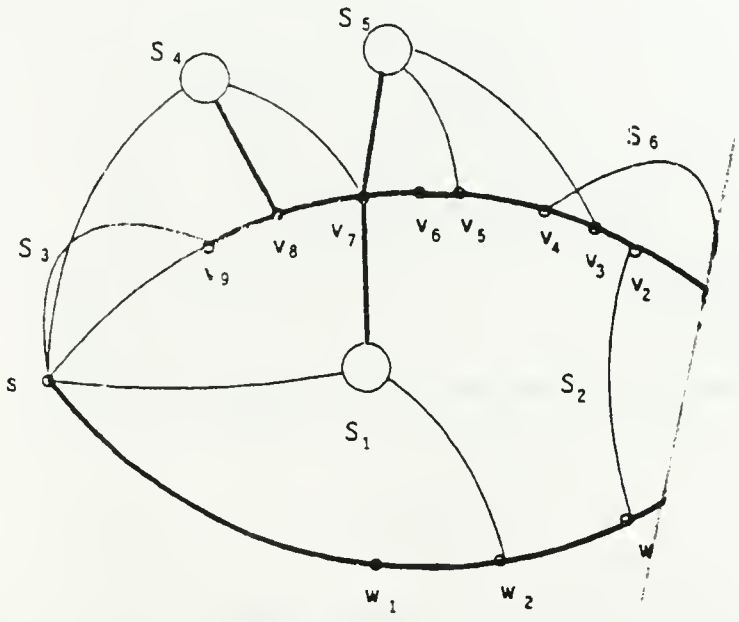


Figure 8:

a B^P-bridge. If a B^P-bridge B has a vertex of attachment at s then it is said to be a *Special B^P-bridge*; otherwise, a *Normal B^P-bridge*.

If B is B^P-bridge then we can also obtain some additional informations about the bridge based on the LOW values of v.

- Let e be a back edge.

  If $v \geq t$ then the left- and right-most vertices of attachment of B are u and v, respectively. In this case, all the edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge*.

  If $v = s$ then the left- and right-most vertices of attachment of B are s and u, respectively. In this case, all but the last edge of the carrier of B are tree edges, B is a *Special B^P-bridge*.

  B is degenerate and hence, has exactly two vertices of attachment.

- Let e be a tree edge.

  If $\text{LOW1}(v) \geq t$ then the left- and right-most vertices of attachment of B are u and $\text{LOW1}(v)$, respectively. In this case, all edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge*.

  If $\text{LOW1}(v) = s$ then the left- and right-most vertices of attachment of B are s and $\text{MIN}(u, \text{LOW2}(v))$, respectively. In this case, all but the last edge of the carrier of B are tree edges, and B is a *Special B^P-bridge*.

  B is proper and if $\text{LOW2}(v) \geq u$ then B has exactly two vertices of attachment; otherwise, B has three or more vertices of attachment.

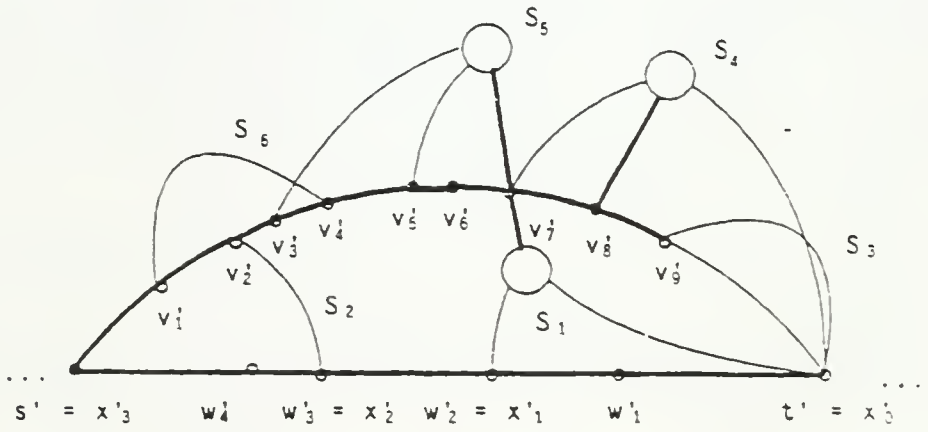


Figure 9:

From the above observations, it is easy to see that the sets $\mathcal{B}^P$ and $\mathcal{V}^P$ can be computed fast. In order to compute the $\mathcal{V}^P$, we simply traverse the edges of the B^{PQ}-bridges.

(2a) The Induction Step: Normal B^P-bridge.

Let B' be a normal B^P-bridge, with its left- and right-most vertices of attachment at s' and t', respectively. Let vertices of attachment of B' be $x'_0, x'_1, \ldots, x'_n$ such that $x'_0 < x'_1 < \cdots < x'_n$, where $n = 1$, if B' is degenerate, and $n \geq 1$, if B is proper.

The carrier of B' can be written as $Q'[x'_0; x'_n] = x'_0 - w'_1 - \cdots - w'_{q-1} - x'_n$, where all the edges of Q' are tree edges, and $x'_0 < w'_1 < \cdots < w'_{q-1} < x'_n$. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$. In this case, either $s' = x'_0$ and $t' = x'_n$, or $s' = x'_n$ and $t' = x'_0$.

- If B' is degenerate then B' is a back edge $x'_1 - x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}(x'_1 \rightarrow x'_0) = x'_1 \rightarrow x'_0$.
- If B' is proper then B' consists of the tree edge $x'_n \rightarrow v'_1$ plus a subtree with root v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$. Let $P'[x'_n; x'_0] = \text{PATH}(x'_n \rightarrow v'_1) = x'_n \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_{p-1} \rightarrow x'_0$, where all but the last edge $v'_{p-1} - x'_0$ are tree edges, and $x'_n < v'_1 < \cdots < v'_{p-1}$. By Lemma 5.2(3) and (4) $= \text{LOW1}(v'_1) = \cdots = \text{LOW1}(v'_{p-1}) = x'_0$.

$J' = \{Q'\} \cup \{P'\}$ is a cycle in $G_{B'}$, consisting of the tree edges $x'_0 \rightarrow w'_1 \rightarrow \cdots \rightarrow w'_{q-1} \rightarrow x'_n - v'_1 \rightarrow \cdots - v'_{p-1}$ and a back edge $v'_{p-1} - x'_0$.

$$x'_0 < w'_1 < \cdots < w'_{q-1} < x'_n < v'_1 < \cdots < v'_{p-1}.$$

Hence the analysis of paths and bridges of $G_{B'}$ can be done in a manner identical to the *basis case*.

(2b) The Induction Step: Special B^P-bridge.

Let B' be a special B^P-bridge, with its left- and right-most vertices of attachment at s' and t', respectively.

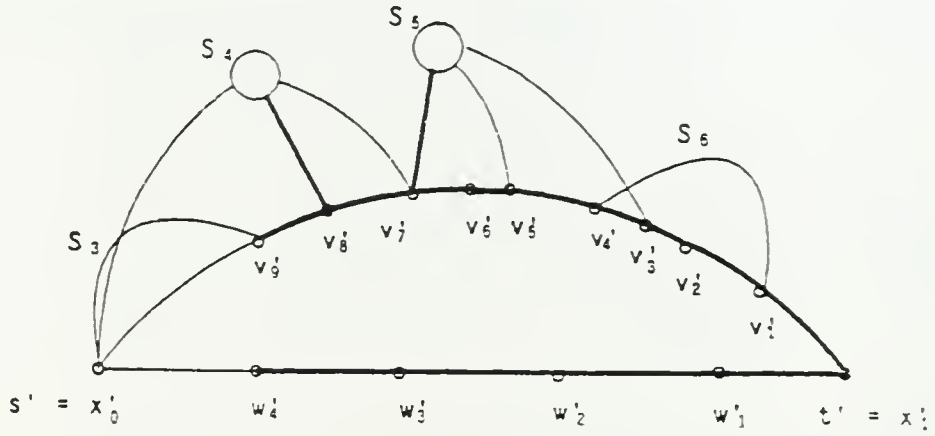


Figure 10:

(CASE1). First, assume that B' *has exactly two vertices of attachment*: x'_0 and x'_1 such that $x'_0 < x'_1$.

The carrier of B' can be written as $Q'[x'_1; x'_0] = x'_1 - w'_1 - \cdots - w'_{q-1} - x'_0$, where all but the last edge $w'_{q-1} - x'_0$ are tree edges, and $x'_0 < x'_1 < w'_1 < \cdots < w'_{q-1}$. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$.

- If B' is degenerate then B' is a back edge $x'_1 - x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}(x'_1 - x'_0) = x'_1 - x'_0$.

- If B' is proper then B' consists of the tree edges $x'_1 - v'_1$ plus a subtree with root v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$. Let $P'[x'_1; x'_0] = \text{PATH}(x'_1 - v'_1) = x'_1 - v'_1 - \cdots - v'_{p-1} - x'_0$, where all but the last edge $v'_{p-1} - x'_0$ are tree edges, and $x'_1 < v'_1 < \cdots < v'_{p-1}$. By Lemma 5.2(3) and (4) $= \text{LOW1}(v'_1) = \ldots = \text{LOW1}(v'_{p-1}) = x'_0$.

$J' = \{Q'\} \cup \{P'\}$ is a cycle in $G_{B'}$, consisting of the tree edges $x'_1 - w'_1 - \cdots - w'_{q-1}$ and $x'_1 - v'_1 - \cdots - v'_{p-1}$, and back edges $w'_{q-1} - x'_0$ and $v'_{p-1} - x'_0$. In this case all the bridges of J' in $G_{B'}$ are B^P-bridges. The analysis of paths and bridges of $G_{B'}$ can be done in a manner similar to the *basis case*.

(CASE2). Next, assume that B' *has three or more vertices of attachment*: x'_0, x'_1, ..., x'_n such that $x'_0 < x'_1 < \cdots < x'_n$. Hence, B' is a proper bridge, with its left- and right-most vertices of attachment at s' and t'.

The carrier of B' can be written as $Q'[x'_1; x'_0] = x'_1 - w'_1 - \cdots - w'_{q-1} - x'_0$, where all but the last edge $w'_{q-1} - x'_0$ are tree edges. Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$. In this case, either $s' = x'_0$ and $t' = x'_1$, or $s' = x'_1$ and $t' = x'_0$—without loss of generality, assume the former.

Hence B' consists of a tree edge $x'_n - v'_1$ plus a subtree with root at v'_1 plus all back edges leading from the subtree. $\text{LOW1}(v'_1) = x'_0$ and $\text{LOW2}(v'_1) = x'_1$. Let $L_1 = \text{PATH}(x'_n - v'_1) =$

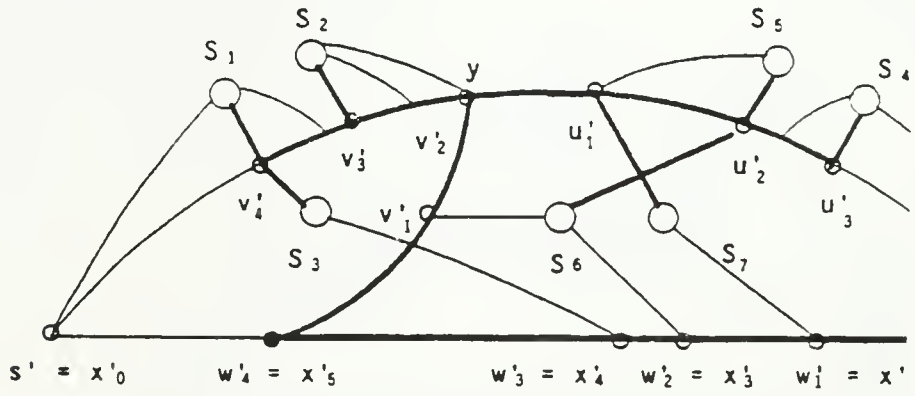


Figure 11:

$x'_n(= v'_0) \rightarrow v'_1 \rightarrow \cdots \rightarrow v'_{m-1} \rightarrow x'_0(= v'_m)$, where all but the last edge $v'_{m-1} \rightarrow x'_0$ are tree edges, and $x'_n < v'_1 < \cdots < v'_{m-1}$. Let

$$y = \text{MAX}(\{v'_i \mid 1 \leq i \leq m-1 \text{ and } \text{LOW2}(v'_i) = x'_1\}).$$

For all $1 \leq j \leq m-1$, if $v'_j \leq y$ then $\text{LOW2}(v'_j) = x'_1$, and if $v'_j > y$ then $\text{LOW2}(v'_j) > x'_1$.

Let S be the segment of L_1 in B' such that one of the following two holds:

- *S consists of a back edge $y \rightarrow x'_1$.* Let $L_2 = \text{PATH}(y \rightarrow x'_1) = y(= u'_0) \rightarrow x'_1(= u'_1)$.
- *S consists of a tree edge $y \rightarrow u'_1$ plus a subtree with root at u'_1 plus all back edges leading from the subtree, and* $\text{LOW1}(u'_1) = x'_1$. Let $L_2 = \text{PATH}(y \rightarrow u'_1) = y(= u'_0) \rightarrow u'_1 \rightarrow \cdots \rightarrow u'_{l-1} \rightarrow x'_1(= u'_l)$, where all but the last edge $u'_{l-1} \rightarrow x'_1$ are tree edges, and $y < u'_1 < \cdots < u'_{l-1}$. $\text{LOW1}(u'_1) = \cdots = \text{LOW1}(u'_{l-1}) = x'_1$.

Let $P'[x'_0; x'_1] = (L_1[y; x'_0])^R * L_2[y; x'_1]$. $J' = \{P'\} \cup \{Q'\}$ is a cycle in $G_{B'}$ and $R' = L_1[x'_n; y]$, a cross-cut of J' between x'_n (an internal vertex of Q') and y (an internal vertex of P'.) $\Theta' = \{J'\} \cup \{R'\}$ is a Θ-graph in $G_{B'}$.

Lemma 5.3 *Let B' be a special bridge with three or more vertices of attachment and L_1, L_2, J' and Θ', as defined earlier. Then*

1. *L_1 and L_2 are well-defined.*
2. *If B is a bridge of Θ' in $G_{B'}$ with a vertex of attachment on $P']x'_0; x'_1[$ then all its vertices of attachment on P' lie either on $P'[x'_0; y]$ or on $P'[y; x'_1]$.*

PROOF.

(1) By Lemma 5.2 L_1 exists. We show that L_2 also exists. Assume to the contrary.

Let $z_1, z_2, \ldots, z_r$ be the adjacency list $A(y)$ of y, ordered by the increasing values of Φ. Since B' has three or more vertices of attachment, $r > 1$. If $y \rightarrow z_1$ is a back edge then $z_1 = x'_0$; and

if $y \rightarrow z_1$ is a tree edge then LOW1(z_1) = x'_0 and LOW2(z_1) > x'_1. Hence, for every $1 < j \leq r$, if $y \rightarrow z_j$ is a back edge then either $z_j = x'_0$ or $z_j > x'_1$, and if $y \rightarrow z_j$ is a tree edge then either LOW1(z_j) = x'_0 and LOW2(z_j) > x'_1 or LOW1(z_j) > x'_1. Hence LOW2(y) > x'_1, thus resulting in a contradiction.

(2) Let B be a bridge of Θ' in $G_{B'}$ with a vertex of attachment on $P']x'_0; x'_1[$. Let the vertices of attachment of B be $x_0, x_1, \ldots, x_n$ such that $x_0 < x_1 < \cdots < x_n$.

Let $x_n = y$. Then B is a segment of L_1 in B' with its highest numbered vertex of attachment at y, and hence, has no vertex of attachment on $L_1]y; x'_0[$ or on $L_2]y; x'_1[$. If B had vertices of attachment at x'_0 and x'_1 then B would consist of a tree edge $y \rightarrow v$ plus a subtree with root at v plus all back edges leading from the subtree, where LOW1(v) = x'_0 and LOW2(v) = x'_1. But by the definition of y, and the DFS order of the palm tree, $\mathcal{P}$, this is impossible.

Let $x_n = v'_j > y$, $(1 \leq j \leq m-1)$. Then B is a segment of L_1 in B' and hence has no vertex of attachment on $L_2]y; x'_1[$. Since LOW2(v'_j) > x'_1, B does not have a vertex of attachment at x'_1.

Let $x_n = u'_j > y$, $(1 \leq j \leq l-1)$. Then B is a segment of L_2 in S and hence has no vertex of attachment on $L_1]y; x'_0[$. Since LOW1(u'_j) = $x'_1 > x'_0$, B does not have a vertex of attachment at x'_0. □

Notice that every bridge of Θ in $G_{B'}$ is either a segment of L_1 in B', other than S, or a segment of L_2 in S. Let u be a vertex on $P']x'_0; x'_1[$, and $e = u \rightarrow v$ be an edge, not belonging to P'. Let B the bridge of Θ in $G_{B'}$, equal to the segment associated with e.

If e is a back edge and $x'_1 < v < y$ then B is a degenerate B^{PQR}-bridge. If e is a tree edge and $x'_1 <$ LOW1(v) $< y$, or $x'_1<$ LOW2(v) $< y$ then B is a proper B^{PQR}-bridge. Otherwise, B is a B^P-bridge. If a B^P-bridge B has a vertex of attachment at x'_0 or at x'_1 then it is said to be a *Special B^P-bridge*; otherwise, a *Normal B^P-bridge*.

If B is B^P-bridge then we can also obtain some additional informations about the bridge based on the LOW values of v.

- Let e be a back edge.

  If $u = v'_j > y$ $(1 \leq j \leq m-1)$ and $v \geq y$ then the left- and right-most vertices of attachment of B are u and v, respectively. If $u = u'_j > y$ $(1 \leq j \leq l-1)$ and $v \geq y$ then the left- and right-most vertices of attachment of B are v and u, respectively. In either case, all the edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge*.

  If $u = v'_j \geq y$ $(0 \leq j \leq m-1)$ and $v = x'_0$ then the left- and right-most vertices of attachment of B are x'_0 and u, respectively. If $u = u'_j \geq y$ $(0 \leq j \leq l-1)$ and $v = x'_1$ then the left- and right-most vertices of attachment of B are u and x'_1. In either case, all but the last edge of the carrier of B are tree edges, B is a *Special B^P-bridge*.

  B is degenerate and hence, has exactly two vertices of attachment.

- Let e be a tree edge.

  If $u = v'_j > y$ $(1 \leq j \leq m-1)$ and LOW1(v) $\geq y$ then the left- and right-most vertices of attachment of B are u and LOW1(v), respectively. If $u = u'_j > y$ $(1 \leq j \leq l-1)$ and LOW1(v) $\geq y$ then the left- and right-most vertices of attachment of B are LOW1(v) and u, respectively. In either case, all the edges of the carrier of B are tree edges, and B is a *Normal B^P-bridge*.

If $u = v'_j \geq y$ $(0 \leq j \leq m-1)$ and $\text{LOW1}(v) = x'_0$ then the left- and right-most vertices of attachment of B are x'_0 and $\text{MIN}(u, \text{LOW2}(v))$, respectively. If $u = u'_j \geq y$ $(0 \leq j \leq l-1)$ and $\text{LOW1}(v) = x'_1$ then the left- and right-most vertices of attachment of B are $\text{MIN}(u, \text{LOW2}(v))$ and x'_1. In either case, all but the last edge of the carrier of B are tree edges, B is a *Special* B^P*-bridge*.

B is proper and if $\text{LOW2}(v) \geq u$ then B has exactly two vertices of attachment; otherwise, B has three or more vertices of attachment.

From the above observations and the Definition 5.2, it is easy to see that the sets $\mathcal{B}^P$ and $\mathcal{V}^P$ can be computed in linear time. In order to compute the $\mathcal{V}^P_{\mathcal{O}}$, we simply traverse the edges of the B^{PQR}-bridges, with a vertex of attachment on $P]s;t[$.

6 Implementation: The Auxiliary Data Structure

In this section we present a data structure for the set $\mathcal{B}^L$ and $\mathcal{V}^L$, generated at each recursive call to the algorithm ANALYZE-BRIDGES. The main function of the data structure is to allow fast implementation of the steps FIND-AND-UPDATE of the algorithm ANALYZE-BRIDGES.

6.1 The Data Structure

Assume that we are given the path L, the sets: $\mathcal{B}^L$ and $\mathcal{V}^L$, and for each bridge B of $\mathcal{B}^L$, its left- and right-most vertices of attachment, and whether it has three or more vertices of attachment. We describe a data structure that supports the following operation:

6.1 THE OPERATION SUPPORTED BY THE DATA STRUCTURE.

- **Find-and-Update:**
  Choose a *Pair* $\langle v, B \rangle$, where $v \in \mathcal{V}'$ and $B =$ an unmarked bridge, maximally covering v, if such a bridge exists; $B = \perp$, otherwise. Remove v from $\mathcal{V}'$.
  Let $\mathcal{B}' \subseteq \mathcal{B}^L$ be the set of bridges, whose carriers intersect with the subpath $L]s_L(B);t_L(B)[$. An unmarked bridge in $\mathcal{B}' \setminus \{B\}$ has a vertex of attachment on $L]s_L(B);t_L(B)[$; add its vertices of attachment to $\mathcal{V}'$. Mark all the unmarked bridges of $\mathcal{B}'$.

Initially, the data structure contains the sets: $V(L[s;t])$, $\mathcal{V}^L$ and $\mathcal{B}^L$, with all the bridges of $\mathcal{B}^L$ *unmarked*. □

We define the *one-dimensional closed* (*integral*) *interval* from x to y,

$$[x;y] = \{i \in \mathbf{N} \mid x \leq i \leq y\}, \quad \text{where } x, y \in \mathbf{N} \text{ and } x \leq y,$$

as the set of all integers between the integer x and the integer y, including x and y. Similarly, the *open-closed*, *closed-open* and *open intervals* from x to y, are defined as:

$$\begin{aligned} \textit{open-closed interval} \text{ from } x \text{ to } y &=]x;y] = \{i \in \mathbf{N} \mid x < i \leq y\}, \\ \textit{closed-open interval} \text{ from } x \text{ to } y &= [x;y[= \{i \in \mathbf{N} \mid x \leq i < y\} \end{aligned}$$

and

$$\textit{open interval} \text{ from } x \text{ to } y =]x;y[= \{i \in \mathbf{N} \mid x < i < y\}.$$

Henceforth, the term *interval* collectively refers to all such intervals, since the ambiguity can be easily resolved by our notation.

An interval $[x_i; y_i]$ is a *subinterval* of the interval $[x_j; y_j]$, *i.e.*, $[x_i; y_i] \subseteq [x_j; y_j]$, if $x_i \geq x_j$ and $y_i \leq y_j$, and a *strict subinterval*, *i.e.*, $[x_i; y_i] \subset [x_j; y_j]$, if either of the inequalities is strict. The subinterval relation is extended to half-open or open intervals in a natural way.

Hence, any closed interval $[x; y] \subseteq [1; k]$ can be visualized as a grid-point (x, y) in a $k \times k$ square, $[1; k] \times [1; k]$. Because $x \leq y$, this *representative point* must lie on or above and to the left of the diagonal line $x = y$. A set of subintervals of $[1; k]$ can thus be visualized as a set of points in the upper left half triangular region, $\{(x, y) \mid 1 \leq x \leq y \leq k\}$, of the $k \times k$ square.

Let $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$ be two distinct points in the upper left half region excluding the diagonal line. Then

$$p_i \succ p_j \ (\text{or } p_j \prec p_i), \qquad \text{if } x_i > x_j, \text{ or } x_i = x_j \text{ and } y_i < y_j.$$

The relation $\succ$ defines a linear order. Let $P = \{p_1 = (x_1, y_1), p_2 = (x_2, y_2), \ldots, p_n = (x_n, y_n)\}$ be a sequence of n distinct points in the upper left half region excluding the diagonal line, ordered by the $\succ$ relation.

1. The point $p_i \in P$ covers a point (u, u) on the diagonal line, if $x_i < u$ and $y_i > u$; and p_i *immediately covers* (u, u), if, in addition, no $p_l \in P$ such that $p_l \succ p_i$, covers (u, u).

2. The point $p_i \in P$ *covers* a point $p_j \in P$, if $x_i \leq x_j$ and $y_i \geq y_j$; and p_i *immediately covers* p_j, if, in addition, no $p_l \in P$ such that $p_l \succ p_i$, covers p_j.

3. The point $p_i \in P$ *l-interlaces* (*i.e.*, interlaces from left) with the point $p_j \in P$, if $x_i < x_j$ and $x_j < y_i < y_j$; and p_i *immediately l-interlaces* with p_j, if in addition, p_i l-interlaces with no $p_l \in P$ such that $p_l \prec p_j$.

The *immediate cover* relation defines a forest structure among the points of P and the points of the diagonal line; and $\text{FATHER}(q) = p$ in the trees, if p immediately covers q.

Also, with each point $p \in P$, we associate a rectangular region, $R(p)$ as follows:

$$R(p) = \begin{cases} [x(q); k] \times [x(q); k], & \text{if } q \text{ immediately covers } p; \\ [1; k] \times [1; k], & \text{Otherwise.} \end{cases}$$

6.2 THE SCHEMATIC REPRESENTATION OF THE DATA STRUCTURE.

Let $v_1 (= s), v_2, \ldots, v_k (= t)$ be the sequence of vertices of the path $L[v_1; v_k]$, ordered from left to right. From now on, we assume that each vertex, v_u is identified by its index u. Hence the vertex u' is *to the left of* the vertex u'', if $u' \leq u''$, and u', *strictly to the left of* u'', if the inequality is strict. Hence, the path L can be represented by the interval $[1; k]$, and a subpath $L[x; y]$ $(1 \leq x \leq y \leq k)$ of L, by the interval $[x; y]$.

- $L[s; t]$: Each vertex u of L has its *representative point* (u, u) on the diagonal line. *This representation can be easily built in time linear in* $|L[s; t]|$, *the length of the path.*
- $\mathcal{B}^L$: Let the set of bridges $\mathcal{B}^L$ be partitioned into the classes, $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n$ where each class contains the bridges with same carrier on L. Each such set of bridges $\mathbf{B}_i \subseteq \mathcal{B}^L$, with

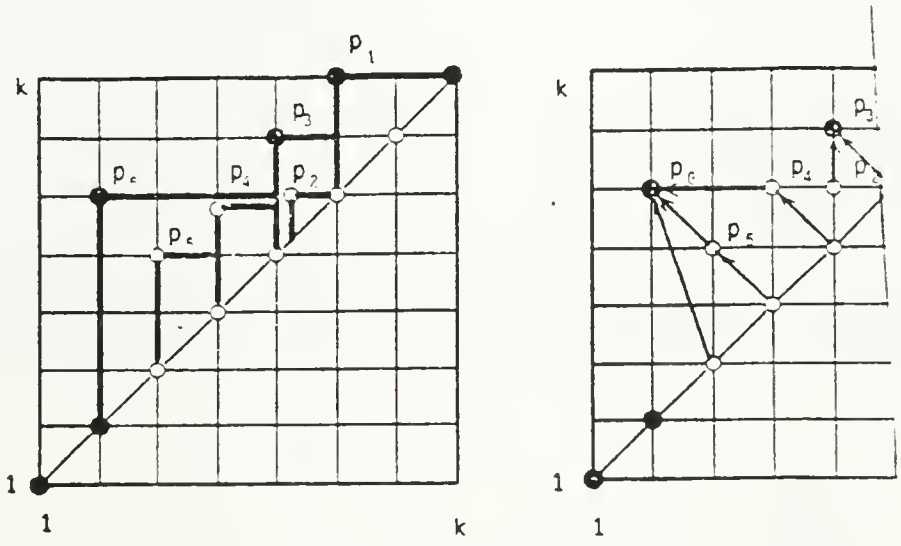


Figure 12:

the same carrier $L[x_i; y_i]$, $(x_i < y_i)$, has its *representative point* $p_i = (x_i, y_i)$ above and to the left of the diagonal line. We also say, $x(p_i) = x_i$, $y(p_i) = y_i$ and $\mathbf{B}(p_i) = \mathbf{B}_i$. Let $P = \{p_1, p_2, \ldots, p_n\}$ be the set of representative points of $\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n$, ordered by the $\succ$ relation. With each point p_i, there is a flag MARK, which is TRUE, if *all* the bridges of $\mathbf{B}(p_i)$ are marked; otherwise FALSE. Initially, all the flags have the truth value FALSE.
Since we know the left- and right-most vertices of attachment of every bridge $B \in \mathcal{B}^L$, this representation can be obtained in time linear in $|L[s;t]| + \left|\mathcal{B}^L\right|$. We use stable radix sort to achieve this bound.

1. The forest F is the structure induced by the immediate cover relation over the points on the diagonal line and the points of P.

   Associated with each point $p \in P$, there is a sequence, *left*(p), of points $p'_1, p'_2, \ldots, p'_m$ immediately l-interlacing with p such that for all $1 \leq i < m$, p'_i immediately covers p'_{i+1}. Further, if q immediately covers p then q also immediately covers p'_1.

2. Associated with each point (u, u), there is a set, *above*$((u, u)) = \{p'_1, p'_2, \ldots, p'_m\}$ such that $u = x(p'_1) = x(p'_2) = \cdots = x(p'_m)$.

   The sequences *left* and *above* are implemented by *endogenous single linear lists*, and hence allow *insertion in the front* in $O(1)$ time and *scanning* of the elements in order in $O(1)$ time per element scanned.

In the next subsection, we describe how to build the forest structure, of (1) *in time linear in* $(|P| + |L[s;t]|)$. *The representation given in* (2) *can easily be built in time linear in* $(|P| + |L[s;t]|)$, *using a stable radix sort algorithm. Since* $|P| \leq \left|\mathcal{B}^L\right|$, *the representation for* $\mathcal{B}$ *can be built in* $O(|L[s;t]| + \left|\mathcal{B}^L\right|)$ *time.*

- $\mathcal{V}'$: The set of vertices in $\mathcal{V}'$ is represented by a *stack* $\mathcal{S}$ and a *bag* $\mathcal{T}$ such that $\mathcal{S} \cup \mathcal{T} = \mathcal{V}'$, $\mathcal{S} \cap \mathcal{T} = \emptyset$. The elements of the stack $\mathcal{S}$ are in a nondecreasing order, the TOP element being a smallest. The elements of the bag $\mathcal{T}$ are unordered. Initially, the stack $\mathcal{S}$ is empty, and the bag $\mathcal{T}$ contains exactly the vertices of initial $\mathcal{V}'$.
  For the *bag* we use an *endogenous single linear list*, which allows both *insertion* and *deletion* in $O(1)$ time. *The time taken to obtain this representation is linear in* $\left|\mathcal{V}^L\right| \leq |L[s;t]|$.

Hence this data structure can be built in time linear in $(|L| + \left|\mathcal{V}^L\right| + \left|\mathcal{B}^L\right|)$. □

6.2 Building the Forest structure

The following algorithm BUILD-FOREST builds the forest structure described in the last subsection and is based on the scan-line paradigm.

```
Algorithm BUILD-FOREST(k, P):
begin
    for u := 1 to k loop  FATHER((u,u)) := nil ;
    for i := 1 to n loop
        FATHER(p_i) := nil ;
        left(p_i) := [ ];
    end{loop };

    Stack := [ ];
    for i := 1 to n loop
        low := x(p_i) + 1;

        while Stack ≠ [ ] cand y(p_i) ≥ y(TOP(Stack)) loop
            p := TOP(Stack);
            high := x(p);
            for u := low to high loop  FATHER((u,u)) := p_i;
            FATHER(p) := p_i;
            low := y(p);
            POP(stack);
        end{loop };

        if Stack = [ ] cor y(p_i) ≤ x(TOP(Stack)) then
            high := y(p_i) - 1;
        else
            high := x(TOP(Stack));
            left(p) := [p_i] & left(p); (* Insert p_i in front of left(p) *)
        end{if };

        for u := low to high loop  FATHER((u,u)) := p_i;
        PUSH(p_i, Stack);
    end{loop };
end{BUILD-FOREST}  □
```

Lemma 6.1 *The algorithm* BUILD-FOREST *correctly builds the forest structure in time* $O(|L[s;t]| + |P|)$.

PROOF.

Let $Stack^0 = [\,]$.

If $Stack^{i-1} = [q'_m, \ldots, q'_j, q'_{j-1}, \ldots, q'_1]$ then $Stack^i = [p_i, q'_{j-1}, \ldots, q'_1] = [q_j, q_{j-1}, \ldots, q_1]$, where $y(q'_m), \ldots, y(q'_j) \leq y(p_i)$ and $y(q'_{j-1}) > y(p_i)$.

We show by induction on i that

1. $\text{TOP}(Stack^i) = p_i$; $x(q_j) < x(q_{j-1}) < \cdots < x(q_1)$; and $y(q_j) < y(q_{j-1}) < \cdots < y(q_1)$.
2. If p_l covers a point in $stack^i$ then $p_l \prec q_j$. $(q_j = p_i = \text{TOP}(Stack^i))$.
3. Let $Stack^i = [q_j, q_{j-1}, \ldots, q_1]$. For all $j \leq l < 1$, if q_l l-interlaces with q_{l-1} then q_l immediately l-interlaces with q_{l-1}.

(1) $\text{TOP}(Stack^i) = p_i$. The rest follows from the inductive hypothesis and the facts that $y(p_i) < y(q'_{j-1})$, and that since $p_i \prec q'_{j-1}$ and $y(p_i) < y(q'_{j-1})$, $x(p_i) < x(q'_{j-1})$.

(2) By the inductive hypothesis, if p_l covers one of $q_{j-1}, \ldots, q_1$ then p_l covers a point of $Stack^{i-1}$, and $p_l \prec p_{i-1}$. But since $y(p_i) < y(q_{j-1})$, $p_l \neq p_i$ and $p_l \prec p_i$, *i.e.*, $p_l \prec q_j$. Moreover, if p_l covers q_j then clearly $p_l \prec q_j$.

(3) Assume that q_j l-interlaces with q_{j-1}. Let $q_j \prec p_l \prec q_{j-1}$. Then p_l is not covered by $q_{j-1}, \ldots, q_1$, and from (2) it follows that q_j covers p_l, and q_j does not l-interlace with p_l. Let $p_l \prec q_j$ then again q_j does not l-interlace with p_l. Hence, q_j l-interlaces with no $p_l \prec q_{j-1}$. The rest follows from the inductive hypothesis.

(*a*) q_j immediately covers (u, u)
iff $x(q_j) < u < y(q_j)$ and no $p_l \succ q_j$ covers u
iff $x(q_j) + 1 \leq u \leq y(q_j) - 1$ and, for no $p_l \in Stack^{i-1}$, $x(p_l) < u < y(p_l)$
iff $\text{FATHER}((u, u)) = q_j$.

(*b*) q_j immediately covers q'_l
iff q_j covers q'_l and no $p_l \succ q_j$ covers q'_l
iff $q_j \prec q'_l$, $y(q_j) \geq y(q'_l)$ and $q'_l \in Stack^{i-1}$
iff $q'_l \in Stack^{i-1}$ and $j \leq l \leq m$
iff $\text{FATHER}(p_j) = p_i$.

(*c*) q_j immediately l-interlaces with q_{j-1}
iff q_j l-interlaces with q_{j-1} and $Stack^i = [q_j, q_{j-1}, \ldots, q_1]$
iff $q_j \in left(q_{j-1})$.

From the order in which the points are examined, it is easy to see that the points of the list, *left*, are ordered by the $\prec$ relation. Also, notice that q_j immediately covers q'_l $(m \leq l \leq j)$ and if $\text{FIRST}(left(q'_l))$ is defined then it is q'_{l+1} and is immediately covered by q_j. Similarly, if q_j immediately l-interlaces with q'_{j-1}, we see that if $\text{NEXT}(q_j, left(q'_{j-1}))$ is defined then it is q'_j and is immediately covered by q_j.

The linearity of the algorithm follows from the observation that the total number of operations is same as the number of FATHER fields in the forest and the sum of the length of all the *left* lists. □

Let p be a point, and $a \geq x(p)$. The following algorithm LIST-LEFT(p, a) lists all the points of the set $\mathcal{L} = \{q' \in R(p) \mid q' \text{ l-interlaces with } p \text{ and } y(q') > a\}$, ordered by the $\succ$ relation.

The structure *List* in the algorithm is implemented by an *endogenous single circular list* and allows *insertion in the front* and *concatenation* in $O(1)$ time, since we allow concatenation to destroy its inputs.

```
Algorithm LIST-LEFT(p, a):
begin
    List := [ ];
    p' := FIRST(left(p));
    while p' ≠ undefined cand y(p') > a loop
        TempList := LIST-LEFT(p', a);
        TempList := [p'] & TempList; (* Insert p' in front of TempList. *)
        List := TempList & List; (* Concatenate TempList to List. *)
        p' := NEXT(p', left(p));
    end{loop };
    return list;
end{LIST-LEFT} □
```

Lemma 6.2 *Let p be a point and let $a \geq x(p)$. Then the algorithm* LIST-LEFT(p, a) *lists all the points of the set $\mathcal{L} = \{q' \in R(p) \mid q' \text{ l-interlaces with } p \text{ and } y(q') > a\}$, ordered by the $\succ$ relation.*

The algorithm is linear in the number of points listed.

PROOF.

The linearity of the algorithm follows from the representation of the lists and the fact that the total number of insertion and concatenation operations is proportional to the number of points listed.

Let $p'_1 \prec p'_2 \prec \ldots \prec p'_s$ be the prefix of the list $left(p)$ such that $y(p'_i) > a$ $(1 \leq i \leq s)$. Let $\mathcal{L} = \{q'_1, q'_2, \ldots, q'_r\}$, its points, ordered by the $\succ$ relation. Notice that, for every $q'_r \prec q' \prec p$, since q immediately covers p, $y(q') < y(p)$ and since $q' \notin \mathcal{L}$, $y(q') \leq a$. Hence, q'_r immediately l-interlaces with p, and $q'_r = p'_s$.

Thus, $\mathcal{L}$ can be written as follows:

$$\mathcal{L} = [p'_s] \,\&\, \mathcal{L}_s \,\&\, \cdots \,\&\, [p'_1] \,\&\, \mathcal{L}_1,$$

where $\mathcal{L}_1 = \{q' \in \mathcal{L} \mid q' \prec p'_1\}$, and for all $1 < i \leq s$, $\mathcal{L}_i = \{q' \in \mathcal{L} \mid p'_{i-1} \prec q' \prec p'_i\}$; and for all $1 \leq i \leq s$, the points in each $\mathcal{L}_i$ are ordered by the $\succ$ relation.

The correctness follows from the following fact, proven by an inductive argument: For all $1 \leq i \leq s$, $\mathcal{L}_i =$ LIST-LEFT(p'_i, a).

1. If q immediately cover p then q immediately covers p'_1; and if no point covers p then no point covers p'_1. In either case, $R(p) = R(p'_1) \supseteq \mathcal{L}_1$.

2. For all $1 < i \leq s$, since p'_{i-1} immediately covers p'_i, $R(p'_i) = [x(p'_{i-1}); k] \times [x(p'_{i-1}); k] \supseteq \mathcal{L}_i$.

Let $1 \leq i \leq s$. Clearly, LEFT-LIST$(p'_i, a) \subseteq \mathcal{L}_i$. Further, $\mathcal{L}_i \subseteq$ LEFT-LIST(p'_i, a): Let $q' \in \mathcal{L}_i$. Then $q' \in R(p'_i)$. Since q' does not cover p'_i and $q' \prec p'_i$, $a < y(q') < y(p'_i)$, and q' immediately l-interlaces with p'_i. $q' \in$ LEFT-LIST(p'_i, a).

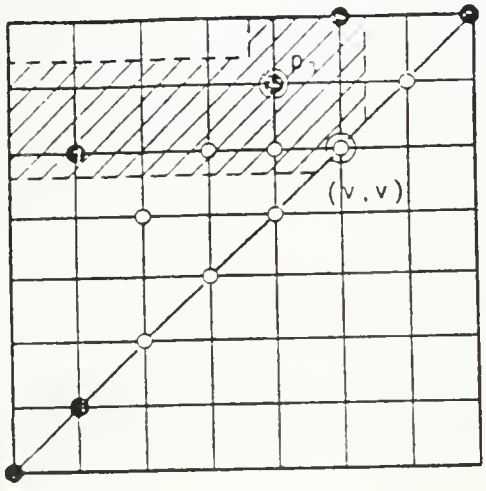
Hence,

$$\mathcal{L} = [p'_s] \,\&\, \text{LIST-LEFT}(p'_s, x) \,\&\, \cdots \,\&\, [p'_1] \,\&\, \text{LIST-LEFT}(p'_1, x),$$

and LIST-LEFT$(p, x) = \mathcal{L}$, the points ordered by the $\succ$ relation. □

6.3 Using the Data Structure

```
Algorithm Find-and-Update:
begin
    if S ≠ [ ] then v := TOP(S); POP(S);
    else Choose any element v from T;   T := T \ {v};
    end{if };
    p' := FATHER(v);
    if p' = nil cor p' = marked then Pair := ⟨v, ⊥⟩;
    else
        p := p';  p' := FATHER(p);
        while p' ≠ nil cand p' = unmarked loop
            p := p';  p' := FATHER(p);
        end{loop }:
        Let B ∈ B(p) be a bridge with exactly two vertices of attachment,
            if such a bridge exists; otherwise, B ∈ B(p), chosen arbitrarily;
        Pair := ⟨v, B⟩;
    end{if };
    if p ≠ ⊥ then
        TempList := LIST-LEFT(p, s_L(B));
        p' := FIRST(TempList);
        while p' ≠ undefined loop
            if p' is unmarked then
                for every B' ∈ B(p') loop
                    (* Let W(B') = vertices of attachment of B'. *)
                    PUSH(s_L(B'), S);
                    T := T ∪ (W(B') \ {s_L(B')});
                end{loop };
                Mark p';
            end{if };
            p' := NEXT(p');
        end{loop };
        for x' := x(p) to y(p) − 1 loop
            for every p' ∈ above((x', x')) loop
                if p' = p then
                    for every B' ∈ B(p') loop
                    (* Let W(B') = vertices of attachment of B'. *)
                    if B' ≠ B cand |W(B')| ≥ 3 then T := T ∪ W(B');
                    end{loop };
                    Mark p';
                elsif p' is unmarked then
                    for every B' ∈ B(p') loop
                        (* Let W(B') = vertices of attachment of B' *)
                        T := T ∪ W(B');
                    end{loop };
                    Mark p';
                end{if };
            end{loop };
        end{loop };
end{Find-and-Update}  □
```



The point, p_3, maximally covers the vertex, v.
All the points in the shaded portion are marked
at the end of the Find-and-Update operation.

Figure 13:

Lemma 6.3 *The algorithm* FIND-AND-UPDATE *correctly implements the operation* FIND-AND-UPDATE *on the data structure.*
PROOF.

First we claim that the stack $\mathcal{S}$ and the points of the data structure satisfy the following two invariants, respectively:

1. The elments of the stack $\mathcal{S}$ are in a nondecreasing order, the TOP element being a smallest.
2. (a) If r is a point marked then all points covering r are also marked.
   (b) If, in addition, $x(r) \notin \mathcal{S}$ then all points that l-interlace with r are marked.

Clearly, the invariant is satisfied initially.

(CASE1). The *pair* $= \langle v, \perp \rangle$. Assume FATHER$((v,v)) = q \neq$ **nil**, since otherwise there is nothing to prove. Hence q must be marked. Since $x(q) < v$, $x(q) \notin \mathcal{S}$. From the invariant on the data structure, it follows that every $p_l \prec q$ covering (v,v) must be marked. But since q immediately covers (v,v) no $p_l \succ q$ covers (v,v). Hence there is no unmarked bridge covering v.

Next we show that, after this operation on the data structure, the invariants are satisfied. The stack obviously satisfies the condition. Hence, assume that the second invariant does not hold. Let r be a point violating the invariant—r is marked. Then it must be the case that $x(r) = v$ and $v \in \mathcal{S}$ just before the FIND-AND-UPDATE operation, and there is an unmarked point l-interlacing with r. But since such a point also covers (v,v), this provides a contradiction.

(CASE2). The *pair* = $\langle v, p \rangle$, where $p = (x, y)$. Assume FATHER$(p) = q \neq$ **nil**, since otherwise there is nothing to prove. Hence q must be marked. Since $x(q) < v$, $x(q) \notin \mathcal{S}$. From the invariant on the data structure, it follows that every $p_l \prec q$ covering p must be marked. But since q immediately covers p no $p_l \succ q$ covers p. Hence there is no unmarked point covering p. Since all the bridges of $\mathbf{B}(p)$ are unmarked, it is easy to see that the bridge $B \in \mathbf{B}(p)$ chosen by the algorithm is in fact the maximal unmarked bridge covering v.

Let us consider a bridge $B' \in \mathcal{B}'$ whose carrier intersects with $L]s_L(B); t_L(B)[$, *i.e.*, the interval $]x; y[$. Hence the representative point, $p' = (x', y')$, of B' has an interval $[x'; y']$ that intersects with $]x; y[$. If $p' \notin R(p)$ then p' either covers or l-interlaces with $q =$ FATHER(p). Thus, p', and hence the bridges of $\mathbf{B}(p')$ must be marked. This implies that we only need to consider such $p' \in R(p)$. Since p is not covered by any unmarked point, we may further exclude the p's covering p from consideration. It is trivial to see that at the end of the operation all such points (hence there bridges) are marked. The correctness of the operation follows from the following observations.

1. $x' < x$. Then if p' is an unmarked point, it is in LIST-LEFT$(p, x(p))$, and p' l-interlaces with p. Hence, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.
2. $x' = x$.
   - $p' = p$. Hence every $B' \in (\mathbf{B}(p) \setminus \{B\})$, with three or more vertices of attachment, is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.
   - $p' \in (\mathit{above}((x, x)) \setminus \{p\})$. Then if p' is unmarked, $y' < y$. Hence, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.
3. $x < x' < y$. $p' \in \mathit{above}((x', x'))$. If p' is unmarked, every $B' \in \mathbf{B}(p')$ is unmarked and has a vertex of attachment on $L]s_L(B); t_L(B)[$. Its vertices of attachment are added to $\mathcal{V}' = \mathcal{S} \cup \mathcal{T}$.

Next we show that, after this operation on the data structure, the invariants are satisfied. Since LIST-LEFT lists the points such that they are ordered by the $\succ$ relation, $x(q'_r) \ldots x(q'_r) \geq x(q'_{r-1}) \ldots x(q'_{r-1}) \geq \cdots \geq x(q'_1) \ldots x(q'_1)$, and thus the stack obviously satisfies the condition. Hence, assume that the second invariant does not hold. Let r be a point violating the invariant—r is marked. We may assume that $r \in R(p)$ and $x(r) < y$. If $x \leq r < y$ then every point covering r or l-interlacing with it is marked at the end of the operation, $x(r) < x < v$. Hence, using the invariant and the fact that $x(r)$ was not in the stack $\mathcal{S}$ before the operation, we see that such an r must have been unmarked before the operation. Hence $r \in$ LIST-LEFT(p); but since at the end of the operation, every point covering r is marked, and $x(r) \in \mathcal{S}$, this provides a contradiction. □

Lemma 6.4 *Let L, $\mathcal{V}^L$ and $\mathcal{B}^L$ be as described in the beginning of this section. Let $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{m'}$ be a sequence of* FIND-AND-UPDATE *operations applied to $\langle L, \mathcal{V}^L, \mathcal{B}^L \rangle$ such that at the end of the m'^{th} operation the set $\mathcal{V}' = \emptyset$.*

Let $\mathcal{Q}' = \langle v'_1, B'_1 \rangle, \langle v'_2, B'_2 \rangle, \ldots, \langle v'_{m'}, B'_{m'} \rangle$ be the sequence of values of the Pair*'s, where $\langle v_i, B_i \rangle$ is the* Pair *returned by the operation $\mathcal{O}_i$. Let $\mathcal{Q} = \langle v_1, B_1 \rangle, \langle v_2, B_2 \rangle, \ldots, \langle v_m, B_m \rangle$ be the subsequence of $\mathcal{Q}'$ consisting of all pairs $\langle v'_i, B'_i \rangle$ such that $B'_i \neq \perp$. Let $\mathcal{B}_1^L = \{B_1, B_2, \ldots, B_m\}$, and $\mathcal{B}_2^L = \mathcal{B}^L \setminus \mathcal{B}_1^L$.*

Then the total time taken to build the initial structure plus the time taken by the operations $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{m'}$ is

$$T\left(L, \mathcal{V}^L, \mathcal{B}^L\right) = O\left(|L| + \left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |E(B)|\right).$$

PROOF.

(1) From the discussion in the previous two subsections, it is clear that the time taken to build the data structure is

$$O\left(|L| + \left|\mathcal{V}^L\right| + \left|\mathcal{B}^L\right|\right).$$

(2) First notice that $m' \leq \left|\mathcal{V}^L\right| + \sum_{B \in \mathcal{B}_2^L} |W(B)|$. Also, notice that if B_i and B_j are two distinct bridges of $\mathcal{Q}$, selected by the operations $\mathcal{O}_i$ and $\mathcal{O}_j$, respectively, then the subpaths $L]s_L(B_i); t_L(B_i)[$ and $L]s_L(B_j); t_L(B_j)[$ do not intersect, and hence $m \leq |L|$. Hence,

$$\left|\mathcal{B}^L\right| \leq \left|\mathcal{B}_1^L\right| + \left|\mathcal{B}_2^L\right| \leq |L| + \left|\mathcal{B}_2^L\right|.$$

Let the *Pair* selected, in the i^{th} operation $\mathcal{O}_i$, be $< v'_i, B'_i >$. Notice that if $B'_i = \perp$ then the amount of time spent in this operation is bounded by constant. On the other hand, if $B_i \neq \perp$ then the amount of time spent in this operation is bounded by the sum of

1. Total number of points in the region

$$([1; t_L(B_i)[\times]s_L(B_i); k]) \setminus ([1; s_L(B_i)[\times]t_L(B_i); k]).$$

2. The time taken to compute the vertices of attachment of those unmarked bridges of $\mathcal{B}' \setminus \{B_i\}$ that have at least one vertex of attachment on $L]s_L(B_i); t_L(B_i)[$. If B is such a bridge then $W(B)$ can be computed in time $O(|E(B)| + |V(B)|) = O(|E(B)|)$ time, using the palm tree of the previous section.

Hence, a bound for the total time spent by the operations $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_{m'}$ can be obtained by deducing bound for the contributions due to (1) and (2). Since any point can belong to at most two different regions of (1), the total contribution due to (1) is bounded by

$$O(|P|) = O\left(\left|\mathcal{B}^L\right|\right).$$

Obviously, the total contribution due to (2) is bounded by

$$O\left(\sum_{B\in\mathcal{B}_2^L}|E(B)|\right).$$

Summing up, we obtain,

$$\begin{aligned}
&O\left(m'+|P|+\textstyle\sum_{B\in\mathcal{B}_2^L}|E(B)|\right)\\
&=O\left(\left|\mathcal{V}^L\right|+|P|+\textstyle\sum_{B\in\mathcal{B}_2^L}|W(B)|+|E(B)|\right)\\
&=O\left(\left|\mathcal{V}^L\right|+\left|\mathcal{B}^L\right|+\textstyle\sum_{B\in\mathcal{B}_2^L}|E(B)|\right).
\end{aligned}$$

Since $\left|\mathcal{B}^L\right|\le|L|+\left|\mathcal{B}_2^L\right|$, we get

$$\begin{aligned}
T\left(L,\mathcal{V}^L,\mathcal{B}^L\right) &= O\left(|L|+\left|\mathcal{V}^L\right|+\left|\mathcal{B}^L\right|+\textstyle\sum_{B\in\mathcal{B}_2^L}|E(B)|\right)\\
&= O\left(|L|+\left|\mathcal{V}^L\right|+\textstyle\sum_{B\in\mathcal{B}_2^L}|E(B)|\right). \quad \square
\end{aligned}$$

7 The Timing-Analysis of the Algorithm Find-Ambitus

Theorem 7.1 *The algorithm* FIND-AMBITUS *requires* $O(|E|+|V|)$ *time to find the ambitus in a graph,* $G=(E,V)$, *where* G *satisfies the assumption 3.1.*

PROOF.

If L is a path then its subpath containing all the vertices of L except its end vertices, is denoted by $L]\cdot\cdot[$. If v is a vertex then $d(v)$ stands for the local degree of v.

The time taken to test whether the graph G satisfies the assumption $= O(|E|+|V|)$. Let $P[s;t]$ and $Q[s;t]$ be the paths in the graph. If T_P and T_Q bound, respectively, the time required to modify $P[s;t]$ and $Q[s;t]$ such that G has no offensive B^P- or B^Q-bridges then the time complexity of the algorithm is bounded by $O(|E|+|V|)+T_P+T_Q$.

The bound for T_P follows from the following recurrence relations.

Let $\mathcal{P}$ be the palm tree of the graph G. The first path in G is J. Let $\mathcal{S}$ be the segments of J in $\mathcal{P}$, and $\mathcal{S}^P$ be the segments corresponding to B^P-bridges. Let $\mathcal{S}_1^P\subseteq\mathcal{S}^P$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^P=\mathcal{S}^P\setminus\mathcal{S}_1^P$.

Hence, T_P is bounded by the sum of (i) the time taken to build the palm tree, (ii) the time taken to find the path J, together with its segments, (iii) the time taken to build the data structure plus that taken by the operations on the data structure, and (iv) the time spent in each recursive call on the segments of $\mathcal{S}_1^P$.

$$\begin{aligned} T_P &\le C_3 \cdot (|E| + |V|) + k_3 \\ &\quad + C_1 \cdot \left(|J| + \textstyle\sum_{v \in V(P]..[)} |d(v)| + \sum_{S \in (\mathcal{S} \setminus \mathcal{S}^P)} |E(S)| \right) + k_1 \\ &\quad + C_2 \cdot \left(|P| + \left|\mathcal{V}^P\right| + \textstyle\sum_{S \in \mathcal{S}_2^P} |E(S)| \right) + k_2 + \textstyle\sum_{S \in \mathcal{S}_1^P} T(S) \\ &\le C_3 \cdot (|E| + |V|) + k_3 + (C_1 + C_2) \cdot \left(|J| + \textstyle\sum_{v \in V(P]..[)} |d(v)| \right) \\ &\quad + (C_1 + C_2 + 1) \cdot \textstyle\sum_{S \in \mathcal{S} \setminus \mathcal{S}_1^P} |E(S)| + k_1 + k_2 + \textstyle\sum_{S \in \mathcal{S}_1^P} T(S) \end{aligned}$$

Let $G_{B'} = \langle E(B') \cup E(Q') \rangle$ be composed of a bridge B' and its carrier Q'; let s' and t' be the two distinguished vertices in $G_{B'}$. Let the segment S' in $\mathcal{P}$ be the one corresponding to B'.

(CASE1). If (i) B' is a normal bridge, or (ii) B' is a special bridge with exactly two vertices of attachment then the first path L found in S' is $P'[s';t']$. Let $\mathcal{S}$ be the segments of L in S', and $\mathcal{S}^L$, the segments corresponding to B^P-bridges of $\{P'\} \cup \{Q'\}$. Let $\mathcal{S}_1^L \subseteq \mathcal{S}^L$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^L = \mathcal{S}^L \setminus \mathcal{S}_1^L$.

Then $T(S')$ is bounded by the sum of (i) the time taken to find the path L', together with its segments, (ii) the time taken to build the data structure plus that taken by the operations on the data structure, and (iii) the time spent in each recursive call on the segments of $\mathcal{S}_1^P$.

$$\begin{aligned} T(S') &\le C_1 \cdot \left(|L| + \textstyle\sum_{v \in V(L]..[)} |d(v)| + \sum_{S \in (\mathcal{S} \setminus \mathcal{S}^L)} |E(S)| \right) + k_1 \\ &\quad + C_2 \cdot \left(|L| + \left|\mathcal{V}^L\right| + \textstyle\sum_{S \in \mathcal{S}_2^L} |E(S)| \right) + k_2 + \textstyle\sum_{S \in \mathcal{S}_1^L} T(S) \\ &\le (C_1 + C_2) \cdot \left(|L| + \textstyle\sum_{v \in V(L]..[)} |d(v)| \right) \\ &\quad + (C_1 + C_2 + 1) \cdot \textstyle\sum_{S \in \mathcal{S} \setminus \mathcal{S}_2^L} |E(S)| + (k_1 + k_2) + \textstyle\sum_{S \in \mathcal{S}_1^L} T(S). \end{aligned}$$

(CASE2). If B' is a special bridge with three or more vertices of attachment then the paths found in S' are L_1 and L_2, and the path $L = P'[s';t']$ is obtained from L_1 and L_2. Notice that $|L| \le |L_1| + |L_2|$. Let $\mathcal{S}$ be the segments of L_1 and L_2 in S', and $\mathcal{S}^L$, the segments corresponding to B^P-bridges of $\{P'\} \cup \{Q'\}$. Let $\mathcal{S}_1^L \subseteq \mathcal{S}^L$ be the set of segments corresponding to the bridges in the sequence of *Pairs*, $\mathcal{Q}$, found by a sequence of FIND-AND-UPDATE operations, and $\mathcal{S}_2^L = \mathcal{S}^L \setminus \mathcal{S}_1^L$.

$$\begin{aligned} T(S') &\le C_1 \cdot \left(|L_1| + |L_2| + \textstyle\sum_{v \in V(L_1]..[)} |d(v)| + \sum_{v \in V(L_2]..[)} |d(v)| \right) \\ &\qquad + \textstyle\sum_{S \in (\mathcal{S} \setminus \mathcal{S}^L)} |E(S)|) + k_1 \\ &\quad + C_2 \cdot \left(|L| + \left|\mathcal{V}^L\right| + \textstyle\sum_{S \in \mathcal{S}_2^L} |E(S)| \right) + k_2 + \textstyle\sum_{S \in \mathcal{S}_1^L} T(S) \\ &\le (C_1 + C_2) \cdot \left(|L_1| + |L_2| + \textstyle\sum_{v \in V(L_1]..[)} |d(v)| + \sum_{v \in V(L_2]..[)} |d(v)| \right) \\ &\quad + (C_1 + C_2 + 1) \cdot \textstyle\sum_{S \in \mathcal{S} \setminus \mathcal{S}_2^L} |E(S)| + (k_1 + k_2) + \textstyle\sum_{S \in \mathcal{S}_1^L} T(S). \end{aligned}$$

Solving the recurrence relation, it is easy to see that

$$T(S) \leq C_1' \cdot \sum_{L \in \mathcal{L}(S)} \left(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v) \right) + C_2' \cdot |E(S)| + (k_1 + k_2).$$

where $C_1' = C_1 + C_2 + k_1 + k_2$, $C_2' = C_1 + C_2 + 1$ and $\mathcal{L}(S)$ is the set of internally vertex disjoint paths in the segment S. Hence it follows that

$$\begin{aligned} T_P \quad &\leq C_3(|E| + |V|) + k_3 \\ &\quad + C_1' \cdot \textstyle\sum_{L \in \mathcal{L}(\mathcal{P})} (|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v)) + C_2' \cdot |E| + (k_1 + k_2). \end{aligned}$$

Since the paths of the palm tree $\mathcal{P}$ are internally vertex disjoint,

$$\sum_{L \in \mathcal{L}(\mathcal{P})} \left(|L| + \sum_{v \in V(L]\cdot\cdot[)} d(v) \right) \leq 2 \cdot |E| + \sum_{v \in V} d(v) \leq 4 \cdot |E|,$$

and

$$\begin{aligned} &T_P \leq C' \cdot |E| + k', \\ &\quad \text{where } C' = 2 \cdot C_3 + 4 \cdot C_1' + C_2' = 5 \cdot C_1 + 5 \cdot C_2 + 2 \cdot C_3 + 4 \cdot k_1 + 4 \cdot k_2 + 1 \\ &\quad \text{and } k' = k_1 + k_2 + k_3. \end{aligned}$$

Hence $T_P = O(|E|)$ by a similar argument $T_Q = O(|E|)$ and the theorem follows. □

8 Abiding Path and Nonseparating Induced Cycle

In this section, we introduce two problems, closely related to the ambitus-finding problem, and devise linear time algorithms for these problems.

Definition 8.1 ABIDING PATH.

Let $G = (V, E)$ be a graph with two distinct vertices s and t, a path $P[s;t]$ and a set of distinguished vertices $W \subseteq V$. A bridge B of $P[s;t]$ is said to be a *W^+-bridge*, if $W \cap N(B) \neq \emptyset$; otherwise, a *$W^-$-bridge*.

Let a *(W,P)-projection* be the set

$$\Pi = (W \cap V(P)) \cup \bigcup_{B = W^+\text{-bridge}} W(G, B),$$

i.e., the set of vertices on $P[s;t]$, either in W or a vertex of attachment of some W^+-bridge.

A path $P[s;t]$ is said to be an *abiding path* with respect to W, if no W^--bridge covers a vertex of the (W,P)-projection, Π. □

Note that if $P[s;t]$ is an abiding path in G then every W^--bridge avoids every W^+-bridge.

Problem 8.1 ABIDING-PATH-FINDING PROBLEM.

Assume that $G = (V, E)$ is a nonseparable graph containing two distinguished vertices s and t, and a set of distinguished vertices $W \subseteq V$.

Find an abiding path $P[s;t]$ in G with respect to W. □

Let $\widehat{G}$ be the graph obtained from G by including additional vertices $\{s', w', t'\}$ and additional edges

$$\{[s', s], [t, t'], [s', w'], [w', t']\} \cup \{[w, w'] \mid w \in W\}.$$

Let $J = P[s';t'] * [t', w'] * [w', s']$ be an ambitus of the graph $\widehat{G}$, containing s' and t'. We claim that the subpath $P[s;t]$ of $P[s';t']$ is an abiding path in G with respect to W. Assume to the contrary; then there is a W^--bridge B of $P[s;t]$ in G, covering a vertex v in the (W, P)-projection. Thus B is a B^P-bridge in $\widehat{G}$ and v is a vertex of attachment of some B^{PQ}-bridge of $\widehat{G}$, *i.e.*, B is an offensive B^P-bridge of $\widehat{G}$. This contradicts the assumption that J is an ambitus.

Note that $\widehat{G}$ has $|V|+3$ vertices and at most $|E|+|V|+4$ edges. Thus the abiding-path-finding problem has a linear time solution.

Definition 8.2 NONSEPARATING INDUCED CYCLE.

Let $G = (V, E)$ be a graph with a cycle J. J is said to be a *nonseparating induced cycle* of G, if there is at most one proper bridge B of J in G. □

Note that the graph G^- derived from G by deleting the vertices of J and all their incident edges is exactly the connected subgraph $N(B)$, the nucleus of B.

Problem 8.2 NONSEPARATING-INDUCED-CYCLE-FINDING PROBLEM.

Assume that $G = (V, E)$ is a triconnected graph containing a distinguished vertex u and a distinguished edge $[s, t]$.

Find a nonseparating induced cycle J of G such that $[s, t] \in E(J)$ and $u \notin V(J)$. □

Let $\widehat{G}$ be the (nonseparable) graph obtained from G by deleting the edge $[s, t]$. Let $P[s;t]$ be an abiding path in $\widehat{G}$ with respect to the set $\{u\}$. Let $J = P[s;t] * [t, s]$ be a cycle in G. We claim that J is a nonseparating induced cycle of G such that $[s, t] \in E(J)$ and $u \notin V(J)$. Clearly, $[s, t] \in E(J)$. If $u \in V(J)$ then every bridge of $P[s;t]$ in $\widehat{G}$ is a $\{u\}^-$-bridge. Thus no bridge of $P[s;t]$ in $\widehat{G}$ covers u, and removal of u in $\widehat{G}$ disconnects the graph, which is impossible.

Thus there is a proper bridge B of J in G such that $u \in N(B)$. If J has another proper bridge B' then, since B' avoids B in J, and since $w(G, B) > 1$ and $w(G, B') > 1$, the vertices of attachment of B' are all contained in a single residual path L of B, and removal of the end vertices of L in G disconnects the graph, which is impossible.

Note that $\widehat{G}$ has $|V|$ vertices and $|E|-1$ edges. Thus the nonseparating-induced-cycle-finding problem has a linear time solution.

9 Acknowledgement

We wish to acknowledge with gratitude the considerable help and the advice we have received from Profs. E.M. Clarke, R. Kannan and R. Statman, all of Carnegie Mellon.

References

[1] J.Cheriyan and S.N.Maheswari, "Finding Nonseparable Induced Cycles and Independent Spanning Trees in 3-Connected Graphs," Technical Report, Indian Institute of Technology, New Delhi, March 1987.

[2] S.Even, *Graph Algorithms*, Computer Science Press, Maryland 1979.

[3] J.Hopcroft and R.Tarjan, "Dividing a Graph into Triconnected Components," *SIAM Journal of Computing*, **2**, September 1973.

[4] J.Hopcroft and R.Tarjan, "Efficient Planarity Testing," *Journal of Association for Computing Machinery*, **21**, October 1974.

[5] S.V.Krishnan, C.Pandu Rangan and S.Seshadri, "A Simple Linear Algorithm for Finding the Ambitus in a Planar Graph," Technical Report, Department of Computer Science, Indian Institute of Technology, Madras, March 1988.

[6] B.Mishra, "An Efficient Algorithm to find All 'Bidirectional' Edges of an Undirected Graph," *25th Annual Symposium on Foundations of Computer Science*, pp. 207–216, 1984.

[7] B.Mishra, *Some Graph Theoretic Issues in VLSI Design.*, Ph.D. Thesis, Carnegie Mellon, September 1985.

[8] T.Ohtsuki, "The Two Disjoint Path Problem and Wire Routing Design," in *Graph Theory and Algorithms* (Eds. N.Saito, T.Nishizeki), Springer 108, October 1980.

[9] O.Ore, *The Four-Color Problem*, Academic Press, New York • London, 1967.

[10] H. Sachs, *Einführung in die Theorie der endlichen Graphen*, Teil H.B.G. Teubner, Leipzig, 1972.

[11] R.Sundar, *Finding the Abiding Path and its Applications to Graph Algorithms*, B.Tech. Thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Madras, June 1987.

[12] W.T.Tutte, *Connectivity in graphs*, University of Toronto Press, Toronto, 1966.

[13] W.T.Tutte, "Bridges and Hamiltonian circuits in planar graphs," *Aequationes Mathematicae*, **15**, 1977.

[14] W.T.Tutte, *Graph Theory*, Addison-Wesley Publishing Co., Menlo Park, California, 1984.

NYU COMPSCI TR-464
Mishra, Bhubaneswar
A linear time algorithm for
finding an ambitus.
c.1

NYU COMPSCI TR-464
Mishra, Bhubaneswar
A linear time algorithm for
finding an ambitus.
c.1

DATE DUE	BORROWER'S NAME

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142

PRINTED IN U S A